Guia práctica de BASIC del

ZX·81

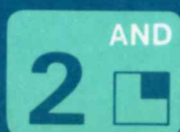
por
RAMON
ROVIRA
SOLIGO

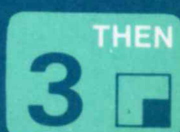
y del Spectrum

- Todas las posibilidades
- Todos los secretos

REVELADOS al USUARIO
























RAMON ROVIRA SOLIGÓ

GUIA PRACTICA DE BASIC DEL ZX-81 Y DEL SPECTRUM

COEDICION

EDICIONES TECNICAS REDE
Apartado 35.400
BARCELONA

VENTAMATIC
Avda. de Rhode, 253
ROSES (Girona)

ISBN 84-247-0190-9

Impreso en España Printed in Spain

D. L.: B. 1853-1984 —REDEPRINT— Barcelona

INDICE

INTRODUCCION

Al lanzar el ZX81 Clive Sinclair ponía la informática al alcance de todos, principalmente en el aspecto económico: por poco dinero cualquier persona podía tener un ordenador en casa. Sin embargo, este objetivo se vio parcialmente truncado a causa del manual suministrado con el ZX81 que, según ciertas críticas, no estaba precisamente al alcance de todos. Personalmente creo que aunque dicho manual tenga bastantes puntos oscuros y trate muy superficialmente cuestiones de gran interés y que han despertado la curiosidad de muchos usuarios, difícilmente podría haberse hecho más completo en el momento del lanzamiento del ZX81, cuando sus mismos autores y diseñadores desconocían gran parte de las posibilidades del aparato que acababan de crear.

El presente libro se publica bastantes meses después de la aparición del ZX81 en el mercado. Ha habido tiempo de aprender mucho sobre él, de descubrir muchos de sus secretos más ocultos, y el resultado es este libro, que no pretende ser más que una alternativa más completa y detallada al manual que se suministra con cada ZX81. Espero que muchos de los puntos que quedaban oscuros en aquél, queden por lo menos parcialmente aclarados en éste, y que pueda servir para que el usuario del ZX81 que lo lea consiga aumentar considerablemente su dominio sobre la máquina y obtener el mejor partido de la misma.

Josep-Oriol Tomás y Huynh-Quan-Chiêu
Coordinador del Club Nacional de Usuarios de los ZX

CAPITULO 1

INTRODUCCION A LOS ORDENADORES

Antes de empezar a trabajar con el ZX81, nos será muy útil conocer los componentes principales de un ordenador.

Podríamos definir brevemente un ordenador como una máquina electrónica que trata unos datos para obtener unos resultados, mediante el uso de una serie de instrucciones. Tanto los datos como las instrucciones se hallan almacenados electrónicamente en lo que se llama memoria.

El conjunto de instrucciones recibe el nombre de programa.

En el texto que sigue, y a modo de ejemplo, se comparará el ordenador con una oficina, en la que hay una secretaria que se encarga de todos los asuntos (fig. 1)

Componentes y estructura

En todo ordenador se diferencian distintos componentes. Veamos cuáles son y las funciones que realizan:

Unidad Central de Proceso o C.P.U.:

La Unidad Central de Proceso se encarga de controlar la ejecución del programa y la producción de resultados. Físicamente es un circuito electrónico susceptible de compararse con el cerebro de nuestra secretaria. Es un cerebro un tanto especial puesto que, al ser un componente electrónico, solamente toma decisiones a través de los impulsos de corriente, al igual que el cerebro responde a los impulsos nerviosos.

El microprocesador distingue en la corriente eléctrica dos estados

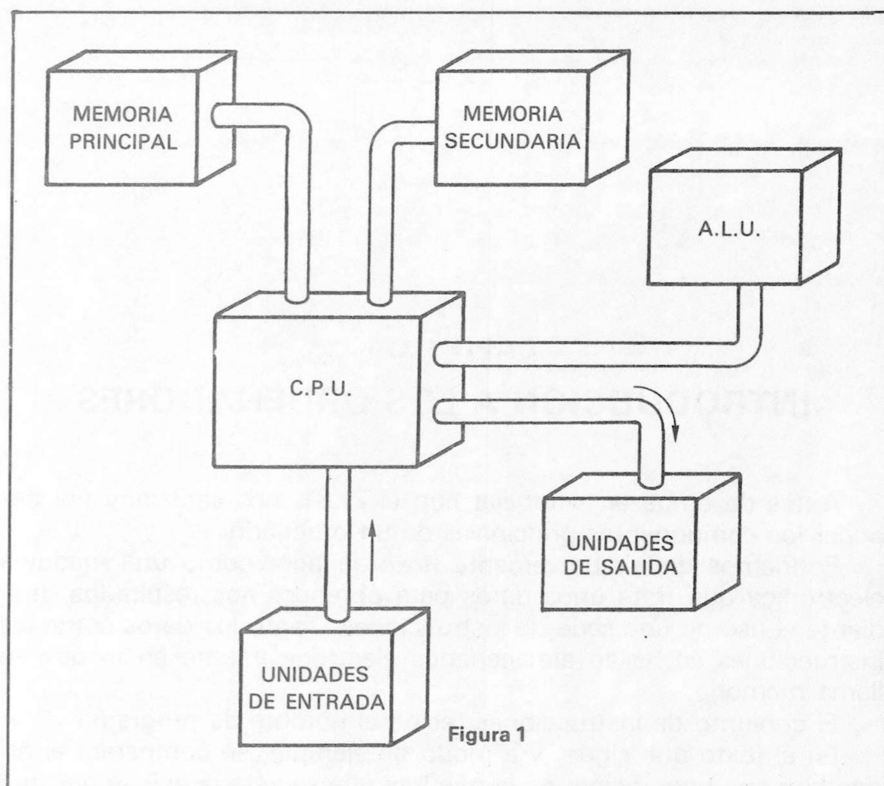


Figura 1

posibles: que exista o que no exista, es decir, la ausencia o presencia de corriente (1).

El bit, unidad mínima de información

La diferencia entre una de estas posibilidades, es la mínima unidad de información posible, que recibe el nombre de *bit* y se representa por un 1 o un 0, según sea el caso (fig. 2).

Esta unidad de información es muy pequeña, puesto que, al microprocesador, con cada orden que le mandemos, sólo se le podrían dar dos tipos de informaciones (2), una para cada valor posible.

---

(1) Hablar de ausencia de corriente es una simplificación, pues en realidad, se trata del potencial de masa.

(2) Se empleará en lo sucesivo, la palabra información para englobar los conceptos de dato e instruccion.

Sería como si a nuestra secretaria, únicamente pudiéramos decirle que colgara el teléfono o que lo descolgara, aunque se lo podríamos repetir cuantas veces quisiéramos. Debido a ello los microprocesadores, al recibir información, la obtienen en grupos de un número determinado de bits.

Cuanto mayor sea este número, más variedad de instrucciones podremos elegir para mandarle cada vez.

En el ejemplo de nuestra secretaria, y en el caso de que «su cerebro fuera de dos bits», sólo podríamos mandarle cuatro instrucciones distintas. Por ejemplo: descolgar el teléfono, coger un bolígrafo, colgar el teléfono y dejar el bolígrafo, representadas por 00, 01, 10 y 11, que son todas las combinaciones posibles con dos (fig. 3).

El microprocesador del ZX81 es un Z80 de ocho bits. Por lo tanto, puede recibir al menos 256 informaciones distintas.

Memoria

El microprocesador por sí solo no puede trabajar, necesita unas instrucciones y unos datos que le son transmitidos, en forma de ceros y unos, desde el lugar en el que están almacenados. A este lugar se le llama *memoria*.

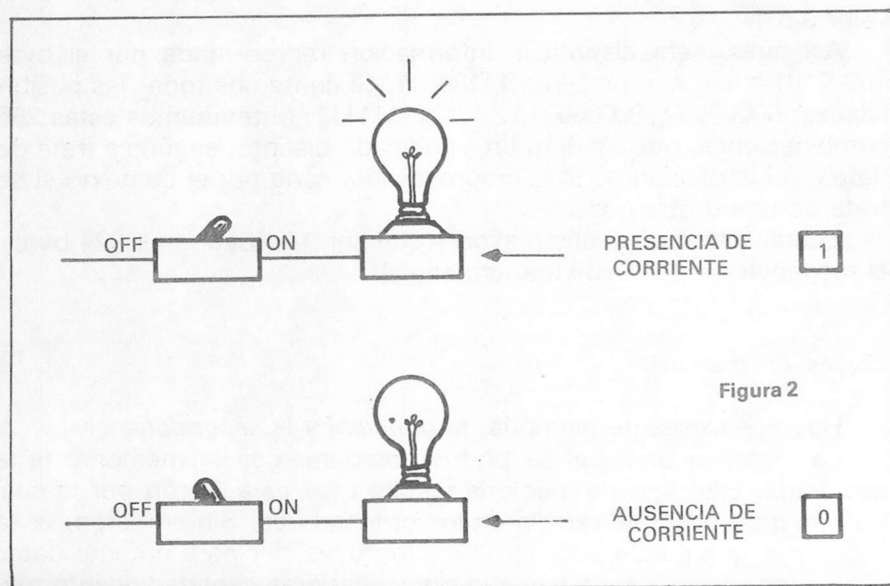


Figura 2

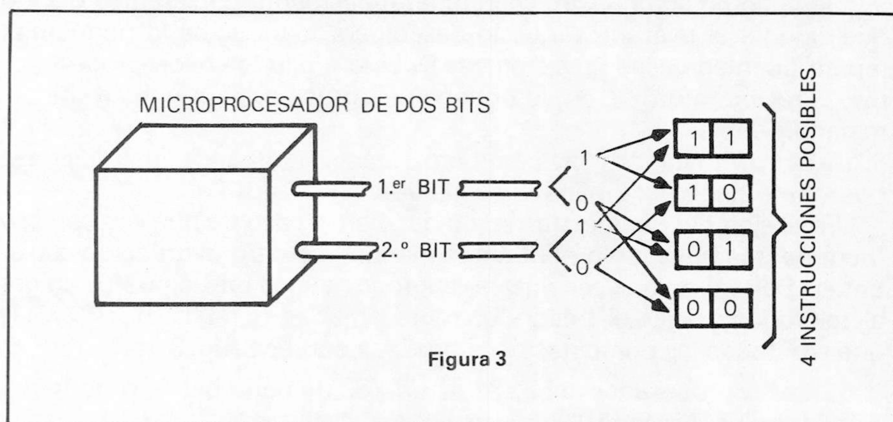


Figura 3

Unidades de memoria

La unidad mínima de memoria es un *bit* y, como ya se ha explicado, sólo puede contener dos clases distintas de información: 1 ó 0, con lo cual no vamos a ninguna parte. Debido a esto, se agrupan en unidades mayores, que son los *bytes.*

Un byte consta de ocho bits. Con él, podemos obtener 256 combinaciones distintas, dado que cada uno de los ocho bits puede tomar el valor 0 ó 1.

Así pues, será distinta la información representada por el byte 10001101 a la que representa 11100011. Si contamos todas las posibilidades: 00000000, 00000001, ... 11111111, obtendremos estas 256 combinaciones, que tendrán un significado distinto, según se trate de datos, o instrucciones. El microprocesador sabe por el contexto si se trata de una u otra cosa.

Existen otras unidades mayores que son el Kbyte = 1.024 bytes; el megabyte = 1.000 Kbytes, etc. (fig. 4).

Clases de memoria

Hay dos clases de memoria: la *principal* y la *secundaria.*

La *memoria principal* se podría comparar con la memoria de la secretaria. Este tipo de memoria resulta muy cara, razón por la cual no hay gran cantidad de ella en los ordenadores. Sin embargo, es la única a la que puede acceder el microprocesador para obtener datos o instrucciones. Cuando hay que almacenar gran cantidad de informa-

ción, se utiliza la *memoria secundaria,* más barata que la principal pero a la que el microprocesador no puede acceder directamente. Cuando se necesita esta información, se introduce en la memoria principal, el microprocesador hace uso de ella y cuando ya no es necesaria es «devuelta» a la memoria secundaria. Dicha memoria secundaria puede ser una cinta magnética, un disco, fichas perforadas, etc.... El ZX81 utiliza cinta magnética como memoria secundaria.

Volviendo a nuestro símil, la memoria secundaria de la secretaria correspondería, por ejemplo, al listín de teléfonos que consulta para marcar un número, a los ficheros de clientes, direcciones, etc....

Tipos de memoria principal

1.ª *ROM:* Es una memoria fija e inalterable y es la que contiene las instrucciones básicas para el funcionamiento del ordenador. En el caso del ZX81, contiene asimismo la interpretación del BASIC.

2.ª *RAM:* Al contrario que la ROM, la información contenida en la RAM puede ser alterada a voluntad, ya sea introduciendo nuevos datos a partir de la memoria externa (secundaria) o a partir del teclado.

En nuestro ejemplo, el contenido de la ROM, correspondería a una serie de acciones que la secretaria tendría que realizar cada día: abrir la puerta, leer la correspondencia, etc... Estas órdenes, tras ser recibidas el primer día, son recordadas permanentemente.

El contenido de la RAM corresponde a la información recibida en

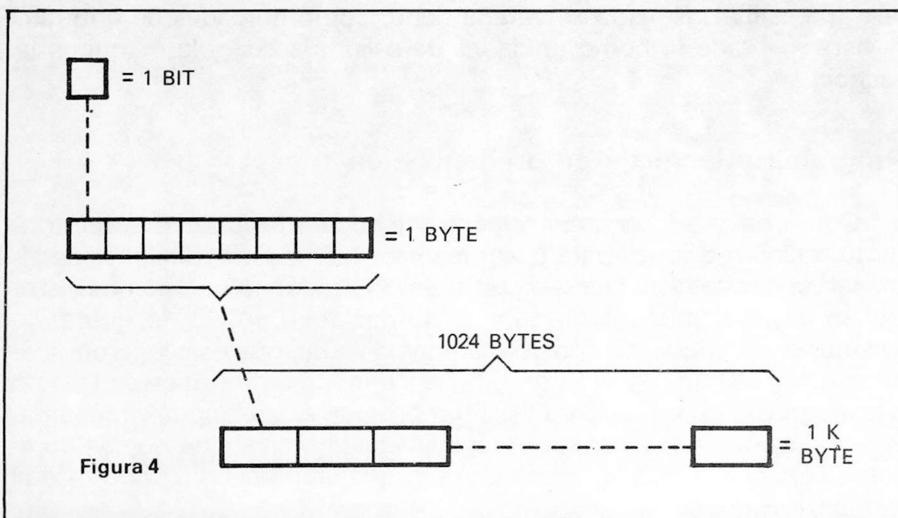


Figura 4

un momento determinado y que es recordada momentáneamente para poder trabajar con ella.

Unidad Aritmética y Lógica

La Unidad Aritmética y Lógica, llamada también A.L.U., es el componente que suma, resta, multiplica y divide (operaciones aritméticas) y también es el que comprueba si un número es positivo, negativo o cero (operaciones lógicas).

En nuestro ejemplo, la Unidad Aritmética y Lógica, correspondería a una calculadora de bolsillo usada por la secretaria para hacer todos los cálculos.

Unidades de entrada y salida

Con los componentes enumerados hasta ahora, un ordenador ya podría trabajar en el caso de disponer de datos e instrucciones pues podría almacenarlos en la memoria, tratarlos con el microprocesador y realizar las operaciones necesarias en la Unidad Aritmética y Lógica. Sin embargo, existiría un problema: no se le podrían dar nuevas instrucciones, ni tampoco se podría ver el resultado de los procesos, con lo cual todo lo anteriormente citado no sería de utilidad alguna.

Las *unidades de entrada y salida* son aquellas que nos permiten obtener resultados (salida) o introducir datos al ordenador (entrada). Las más corrientes son el teclado (de entrada) y la impresora y la pantalla (de salida). Nuestra secretaria tiene, como unidades de entrada, la vista y el oído y, como unidades de salida, la voz y la máquina de escribir.

Programa y lenguaje de programación

Como ha quedado anteriormente dicho, un *programa* es un conjunto de instrucciones para tratar los datos. Estas instrucciones se almacenan por pasos o líneas de programa y, según lo que hemos visto, deben llegar al microprocesador en forma de 1_s y 0_s o, lo que es lo mismo, en el lenguaje o código máquina. Al Z80, que es el microprocesador del ZX81, le llegan en grupos de 8 bits, un byte cada vez. Debido a la laboriosidad de escribir y descifrar programas en lenguaje máquina, se han inventado los llamados lenguajes de alto nivel que no son comprensibles por el microprocesador pero que se parecen mucho más al lenguaje humano.

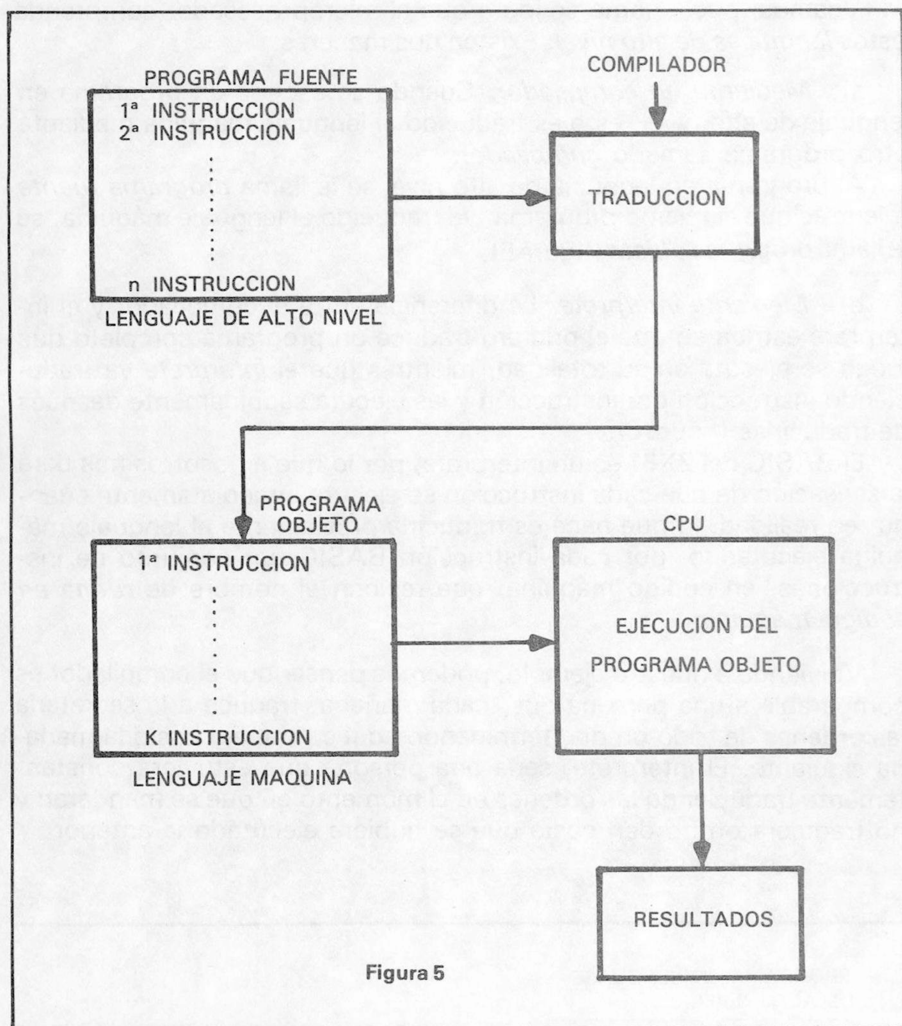


Figura 5

Estos lenguajes de alto nivel son hoy en día variados y numerosos. Ejemplos de ellos son el Fortran, Cobol, BASIC, el RPG, etc., y cada uno de ellos cuenta con infinidad de versiones distintas.

En nuestro ejemplo, esta situación correspondería a que la secretaria sólo entendiera un idioma, por ejemplo, el inglés, el cual equivaldría al lenguaje máquina, y que sus superiores le dieran órdenes en distintos idiomas, que tendrían que ser traducidas para ser ejecutadas, problema éste que trataremos a continuación.

Veamos, pues, cómo se logra que el microprocesador comprenda estos *lenguajes de alto nivel.* Existen dos maneras:

a) *Mediante un compilador.* Cuando se escribe un programa en lenguaje de alto nivel, éste es traducido al lenguaje máquina mediante otro programa llamado *compilador.*

Al programa en lenguaje de alto nivel se le llama *programa fuente* mientras que al mismo programa, ya traducido al lenguaje máquina, se le llama *programa objeto* (figura 5).

b) *Mediante intérprete.* La diferencia entre el compilador y el intérprete estriba en que el primero traduce un programa completo que luego se ejecuta en su totalidad, mientras que el *intérprete* va traduciendo instrucción por instrucción y las ejecuta seguidamente después de traducirlas (figura 6).

El BASIC del ZX81 es un intérprete, por lo que a nosotros nos dará la sensación de que cada instrucción se ejecuta inmediatamente cuando, en realidad, lo que hace es traducirla previamente al lenguaje máquina ejecutando, por cada instrucción BASIC, un conjunto de instrucciones, en código máquina, que reciben el nombre de *rutina en código máquina.*

Volviendo a nuestro ejemplo, podemos pensar que el compilador es comparable a una persona que, cada mañana, traduce a la secretaria las órdenes de todo un día, terminando aquí su trabajo hasta la mañana siguiente. El intérprete, sería una persona que estuviera constantemente traduciendo las órdenes en el momento en que se mandaran y no tradujera otra orden hasta que se hubiera ejecutado la anterior.

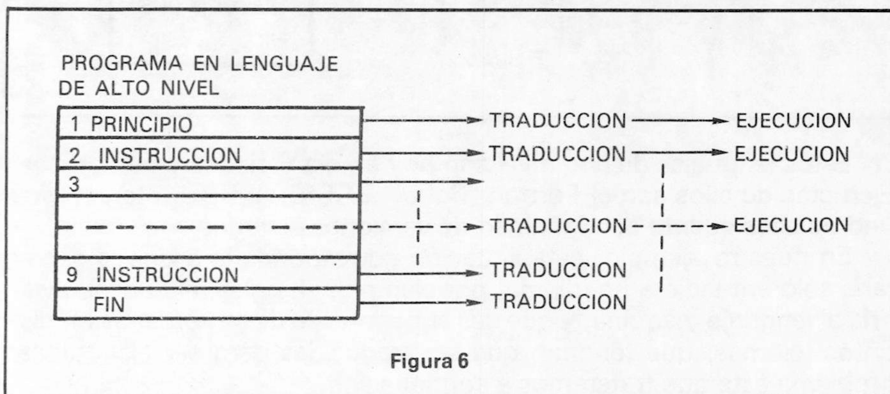


Figura 6

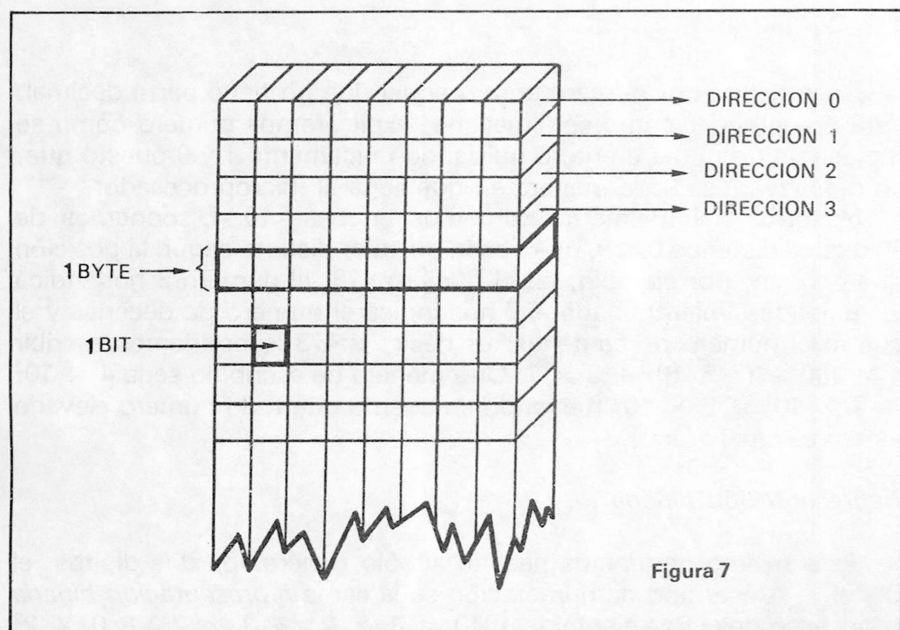


Figura 7

Representación de números en memoria

Aunque vimos que la mínima unidad para representar información, ya sean datos o instrucciones, es un bit, ya se hizo notar anteriormente que el ZX81 recibe la información en grupos de ocho bits (1 byte). Por lo tanto, la mínima unidad de información indivisible es el byte.

A partir de ahora, cuando hablemos de una posición de memoria o de una dirección de memoria, nos estaremos refiriendo a un byte.

Las direcciones de memoria se designan por números enteros que van desde el 0 hasta el número máximo de bytes de memoria disponibles. El Z80 puede llegar a direccionar hasta 65.535 posiciones (figura 7).

El tema de la representación y almacenamiento de números en memoria no es necesario dominarlo de entrada. Está incluido aquí para ser consultado en caso de necesidad y trata específicamente de la representación numérica del ZX81 puesto que en otros ordenadores hay ligeras variaciones.

En general, se distinguen dos tipos de números: los enteros y los reales. Según sea uno u otro tipo, se representarán y almacenarán de manera distinta.

Número entero

Se entiende por *número entero* aquel que no tiene parte decimal. Para comprender cómo se almacena, explicaremos primero cómo se representa un número entero utilizando únicamente 1 y 0 puesto que, en definitiva, esa es la manera en que llega al microprocesador.

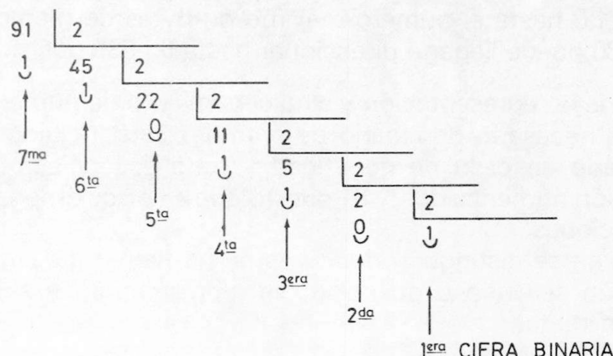
Nosotros habitualmente contamos en base 10. Disponemos de 10 digitos distintos 0... 9, que tienen un valor distinto según la posición que ocupan. Por ejemplo, en el número 473, el número 3 nos indica las unidades, mientras que el 7 nos indica el número de decenas y el cuatro el número de centenas; es decir, el 473 lo podríamos escribir $4 \times 100 + (7 \times 10) + 3 \times 1$. Otra manera de escribirlo sería $4 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$ (teniendo en cuenta que todo número elevado a cero es igual a uno).

Representación binaria

Esto mismo podríamos hacerlo si sólo tuviéramos dos dígitos, el 0 y el 1. A este tipo de numeración se la llama *representación binaria* o en base dos. Por ejemplo $11010 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 16 + 8 + 0 + 2 + 0 = 26$. El número 11010 en base dos corresponde pues al número 26 en base diez.

La operación contraria consiste en pasar un número en base diez a base 2. Esto se realiza dividiendo sucesivamente por 2 y cogiendo como cifras binarias el último cociente y los restos en orden ascendente.

Para saber cómo se representa 91 en base 2, haremos:



Por tanto: $91_{10)} = 1011011_{2)}$ *para que ocupe exactamente un byte, se le añade un 0 delante y queda:*

0	1	0	1	1	0	1	1

Si el número tiene parte fraccionaria se actúa del modo que se explica a continuación. Se separan la parte entera de la parte decimal. La parte entera se convierte a base dos del mismo modo que se ha explicado anteriormente. Y para la parte decimal se realizarán los siguientes pasos: se escribe la parte decimal añadiendo al principio 0 y coma; el número resultante se multiplica por dos y se toma como primera cifra de la parte fraccionaria la parte entera de este resultado que siempre es 1 ó un 0. Seguidamente se le resta dicho número (el 1 ó el 0) y se vuelve a multiplicar el resultado por dos, tomándose como segunda cifra la parte entera y así sucesivamente. Comprobémoslo con un ejemplo:

$0{,}6875_{10)} = ?_{2)}$

$0{,}6875$
$\times 2$
$1{,}3750$ ⟶ *1 es la primera cifra*
$\times 2$
$0{,}750$ ⟶ *0 es la segunda cifra*
$\times 2$
$1{,}50$ ⟶ *1 es la tercera cifra*
$\times 2$
$1{,}0$ ⟶ *1 es la cuarta cifra*

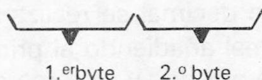
tenemos pues: $0{,}6875_{10)} = 0{,}1011_{2)}$

al revés: $0{,}1011_{2)} = 2^{-1} + 0 + 2^{-3} + 2^{-4} = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} = \frac{11}{16} = 0{,}6875$

De este modo se observa que las cifras binarias de la parte fraccionaria tienen los valores 2^{-1}, 2^{-2}, 2^{-3}, etc.

Veamos ahora cuál es el número mayor que podemos representar en un byte: es aquél en el que las ocho posiciones tengan el valor 1, es decir 11111111 = $2^7 + 2^6 + 2^5 + \ldots + 2^1 + 2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$.

El ZX81 utiliza principalmente los números enteros para designar las posiciones de memoria. Como hay más de 255 posiciones, necesitaremos dos bytes para representarlas todas. Procediendo de igual forma, vemos que el número mayor que podemos representar en dos bytes es el 11111111 11111111 = 65535, que ya es suficiente para de-



signar todas las posiciones de memoria permitidas por el ZX81.

Resumiendo, un número entero de un byte está en la serie 0-255 y un número entero de dos bytes está en la serie 0-65535.

Representación hexadecimal

En ocasiones también se emplea la *representación en base 16 ó hexadecimal* en la que se emplean 16 dígitos, que son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,; donde A, B, C, D, E, F, representan a los números 10, 11, 12, 13, 14, 15, respectivamente.

La razón de que se use la numeración hexadecimal es la que se expone a continuación.

Observemos que el mayor número que se puede representar con dos dígitos es el FF = $15 \times 16 + 15 \times 1 = 255$, es decir, cada byte se corresponde con dos dígitos hexadecimales o, lo que es lo mismo, cada dígito hexadecimal se corresponde a 4 bits. Para el cambio de base decimal a base 16 se procede de igual forma que para pasar a base 2 pero dividiendo y multiplicando por 16 según corresponda a la parte entera o decimal respectivamente. Otra forma de hacerlo consiste en pasar el número previamente a base 2 y añadirle delante los ceros que sean necesarios para que el número de cifras sea un múltiplo de 4. Después, se separa el número binario en grupos de cuatro cifras y se sustituye cada grupo por el dígito hexadecimal que corresponda según la siguiente tabla:

0000	———	0
0001	———	1
0010	———	2
0011	———	3
0100	———	4
0101	———	5
0110	———	6
0111	———	7
1000	———	8
1001	———	9
1010	———	A
1011	———	B
1100	———	C
1101	———	D
1110	———	E
1111	———	F

Número real

Olvidando el concepto puramente matemático de *número real,* lo podemos definir como un número que tiene parte decimal finita, infinita o nula, estando, en este último caso, incluidos los enteros dentro de los reales.

Muchos números reales, concretamente los irracionales y los racionales periódicos, no se pueden almacenar exactamente pues tienen infinitas cifras decimales. Tampoco es posible hacerlo con un número que tenga tantas cifras que no quepan en el espacio de memoria designado para almacenarlas. La precisión de un número viene dada por el número de cifras significativas que posee.

Notación científica

La *notación científica* consiste en escribir el número como producto de sus cifras significativas por 10 elevado a un exponente. Por ejemplo:

$$0{,}2432 \times 10^{4} = 2432$$

$$2432 \times 10^{-4} = 0{,}2432$$

Este tipo de notación adaptada al método de numeración binaria es usada por los ordenadores para almacenar los números, siendo conocido como representación en coma flotante. Más adelante veremos cómo funciona exactamente este tipo de representación en el ZX81. De momento, sólo necesitamos saber que para cada número real se emplean cinco bytes de memoria, siendo el primero el que representa al exponente.

CAPITULO 2

INSTALACION Y MANTENIMIENTO

El ZX81 llega a sus manos en una caja que contiene los siguientes objetos:

—La *unidad básica* que integra principalmente la Unidad Central de Proceso, la Unidad Aritmética y Lógica, 8 K de ROM, 1 K de RAM, el teclado y cinco salidas. La de la parte posterior sirve para conectar ampliaciones de memoria, la impresora y otros muchos dispositivos que se encuentran en el mercado (por ejemplo, placa de alta resolución, placa de sonido, etc.). La salida marcada 9VDC es para alimentación, EAR es por donde salen las señales para el magnetófono. MIC es la entrada para señales procedentes del magnetófono y, finalmente, TV que emite la señal para el televisor (fig. 8).

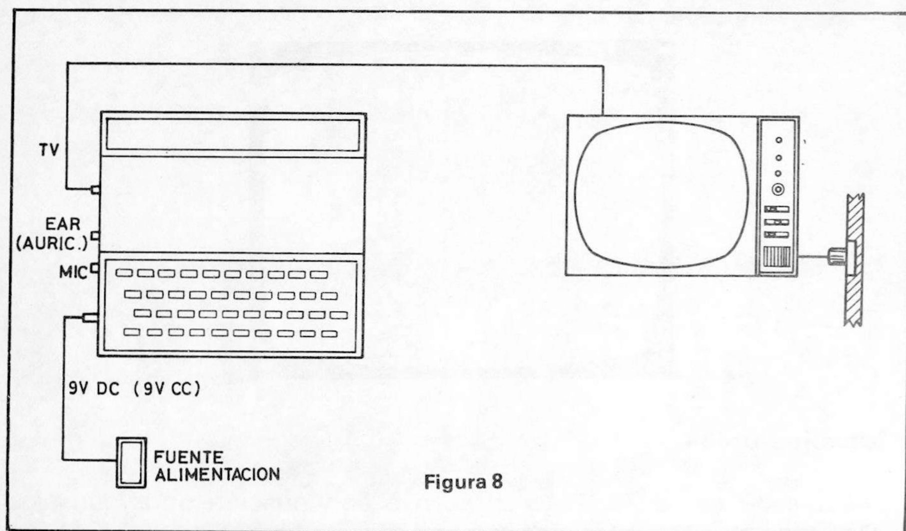


Figura 8

—Un juego de cables, utilizados para las conexiones con el magnetófono.

—Un cable más largo de antena.

—El libro de instrucciones.

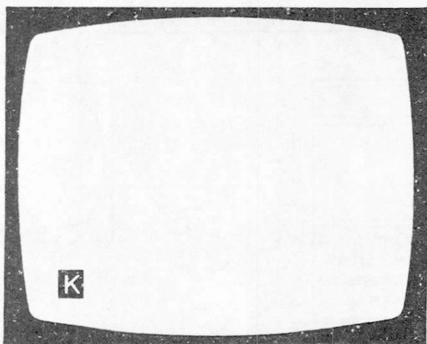
Antes de empezar a trabajar, léase bien estas instrucciones para la conexión e instalación:

a) El cable de antena tiene unos 120 cm de largo. Debe ser conectado a la salida marcada con TV del ZX81 y, el otro extremo, a la entrada de antena de su televisor, que deberá disponer de UHF.

b) La fuente de alimentación va conectada a la toma de corriente y el otro extremo al enchufe marcado 9VDC.

c) Las salidas MIC y EAR, junto con los otros dos cables que se acompañan, sirven para conectar el ZX81 al magnetófono, cosa que veremos más adelante (Capítulo 13). Ahora sólo adelantaremos que si el ZX81 se desconecta pierde toda la información que contiene y el único modo de que no se pierda es registrándola en un magnetófono, que hace así la función de memoria secundaria.

El ZX81 no dispone de interruptores de modo que, al conectarle la fuente de alimentación, queda automáticamente en funcionamiento. Una vez hecho esto y conectado al televisor (que hace la función de unidad de salida junto con la impresora, que no va incluida y que se vende aparte), debe sintonizar éste al canal 36 de UHF. Cuando todo está correcto, ofrece una imagen como esta:



Consejos útiles

Instalado ya su ZX81, se ofrecen a continuación unos consejos útiles para el mejor aprovechamiento de este libro:

1.° Siempre que tenga algún problema puede desconectar y volver a conectar la clavija 9VDC y recobrar así la imagen inicial, aunque a costa de perder toda la información.

2.° Escriba lo que usted escriba no puede dañar al ZX81.

3.° No deje a un lado los ejercicios propuestos ya que le darán una cierta agilidad y le ayudarán a afianzarse en los conceptos ya explicados.

Cuidado y mantenimiento

Cada vez que conecte y desconecte una clavija, no lo haga bruscamente, pues con el tiempo se puede ensanchar la hembra y producir una desconexión nada deseable: la pérdida total de información.

Debido a efectos electroestáticos, la pantalla del televisor se ensucia muy rápidamente. En tal caso, límpiela con un paño húmedo.

Aparte de todo esto, tanto el televisor como el ordenador no requieren otras atenciones mientras que el magnetófono sí que requiere especial atención como ya veremos más adelante (Capítulo 13), incluso en el momento de su elección. Es aconsejable, siempre que sea posible y por razones de seguridad y comodidad, disponer de una instalación fija.

CAPITULO 3

LA FAMILIARIZACION CON EL ZX81

Conocidos ya los principales componentes de un ordenador, vamos a empezar a trabajar con el ZX81.

Una vez conectado el ordenador, tal como se indica en el Capítulo anterior y si lo ha hecho correctamente, le aparecerá en la pantalla la K en inversa, en el extremo inferior izquierdo. Esta letra se conoce con el nombre de *cursor* y nos indica la posición en que se va a imprimir el próximo carácter que pulsemos:

La K, viene del inglés «keyword», que significa palabra clave. Esto nos indica que cuando pulsemos una tecla lo que se va a escribir es una palabra clave, que son las que están escritas encima del teclado.

Estructura del teclado

Al pulsar una tecla, lo que se imprime depende del estado en que se halle el cursor. Este puede encontrarse de cuatro modos distintos que son: K, L, F, G.

Modo K:

Ya hemos visto que cuando el cursor está en modo K, aparecen las palabras claves. También está de este modo cuando permite la introducción de números de línea.

Modo L:

Cuando está en modo L, aparece el carácter principal. Para hacer aparecer los caracteres en rojo se debe pulsar simultáneamente la tecla SHIFT y éstos se imprimirán indistintamente de que el cursor sea K o L.

Modo F:

El modo F indica función. Para que aparezca la F hay que pulsar simultáneamente [SHIFT] y [NEW LINE]. Entonces la próxima tecla que se pulse provocará la impresión de la función correspondiente a esa tecla. Después, el cursor cambiará de nuevo a modo L.

Las funciones son las palabras que están escritas debajo de las teclas.

Modo G:

Para conseguir el cursor en modo G, se tiene que pulsar [SHIFT] y [9]. Entonces los caracteres que se impriman a continuación serán el inverso del que sale en modo L. Si se pulsa simultáneamente [SHIFT] aparece el carácter gráfico o bien el inverso del símbolo en rojo en caso de que no haya carácter gráfico en la tecla.

Las figuras 9 y 10 muestran cómo obtener cada uno de los caracteres de una tecla.

Haga pruebas con esto, para obtener los distintos caracteres de las teclas. Conseguirá de este modo agilidad en el manejo del teclado.

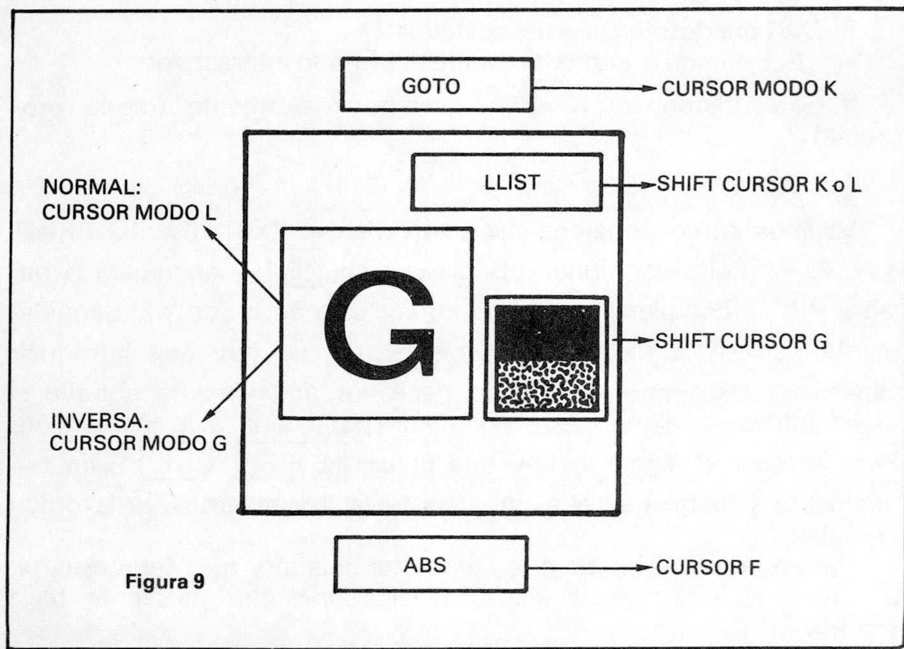


Figura 9

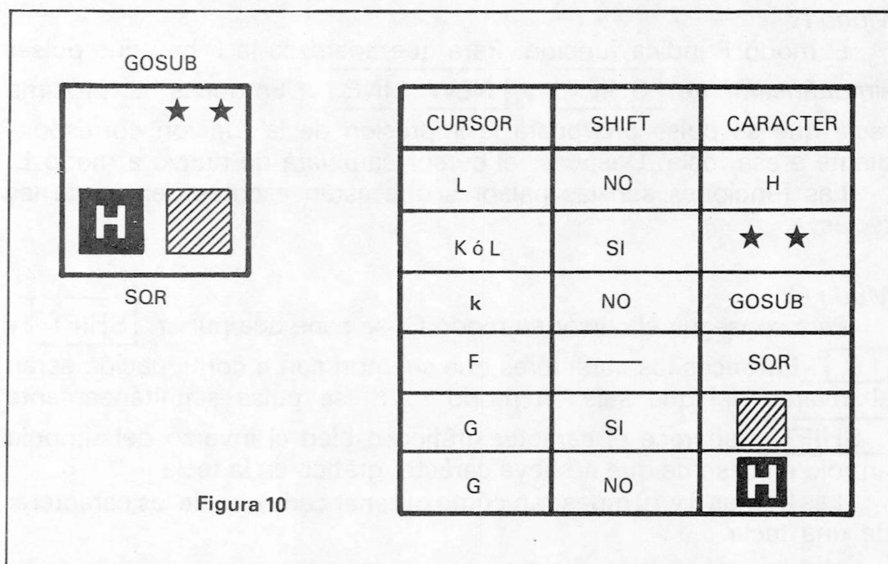


Figura 10

Los dos estados en que puede trabajar el ZX81

El ZX81 puede trabajar en dos estados:

a) Recibiendo órdenes directamente (modo interactivo).

b) Ejecutando un programa escrito previamente (modo programa).

a) *Modo interactivo*

Veamos cómo podemos dar una orden al ZX81. Escriba PRINT 3 + 4. Para ello tiene que pulsarse la tecla [P] y aparecerá la palabra PRINT completa (ya que el cursor está en modo K). Seguidamente púlsese la tecla [3]. Observe que no hay que introducir espacios puesto que el ZX81 lo hace automáticamente aunque si usted introduce alguno éste no afecta para nada a la instrucción. Para obtener el signo + hay que pulsar [SHIFT] y [K] simultáneamente y luego hay que pulsar la tecla 4 y tenemos ya la orden completa.

Para que el ZX81 sepa que hemos terminado y que debe ejecutar la orden que le hemos dado, usted tiene que pulsar la tecla [NEW LINE].

Esta tecla hay que pulsarla cada vez que se termina una orden o se introduce una línea de programa y sólo lo indicaremos las primeras veces.

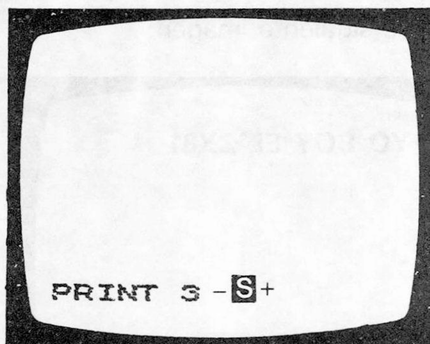
Una vez pulsada la tecla NEW LINE, verá que aparece un 7 en la esquina superior derecha de la pantalla. Esto quiere decir que el ZX81 ha calculado el valor de la expresión 3 + 4, y lo ha impreso tal y como se lo hemos transmitido.

Usted no debe intentar deletrear las palabras clave pues el ordenador no lo entendería. Observará que, aparte del 7, en la parte alta, aparece el informe 0/0 en la esquina inferior izquierda. El primer 0 significa «todo bien», mientras que el segundo nos indica la línea en la que se ha detenido el programa. Como en tal caso lo ejecutado era una orden, a efectos de informes el ordenador toma como número de línea el 0.

Una vez aparecido el informe, podemos introducir otra orden. Pruebe a hacer otras prácticas con la instrucción PRINT y observe cómo el ZX81 cumple sus órdenes.

Errores de sintaxis

Si usted introduce una orden que no tiene sentido, por ejemplo "PRINT 3 — +" cuando pulse NEW LINE obtendrá una imagen como ésta:



La S es el indicador de error de sintaxis y nos advierte de que el ordenador sólo entiende hasta donde se encuentra la S y que el resto de la frase no está correcta. Veamos cómo corregirlo:

1.º Sitúe el cursor justo detrás del carácter a modificar. (Para ello utilice ⇐ y ⇒ (SHIFT 5 y SHIFT 8 respectivamente).

2.º Borre el carácter utilizando RUBOUT (SHIFT 0).

Si en lugar de borrar un carácter, desea añadirlo, sitúe el cursor en el lugar escogido e introduzca el carácter. Esto provoca la aparición del mismo en el lugar que ocupaba el cursor, que se traslada una posición a la derecha.

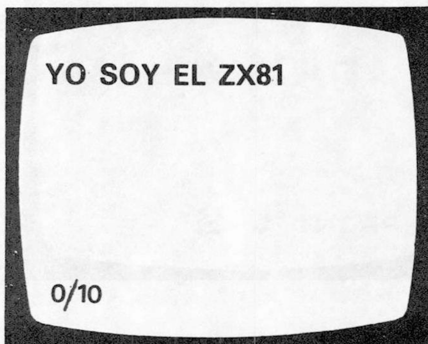
El cursor sólo indica el carácter a borrar o bien el lugar en que se imprimirá el próximo, pero no ocupa ningún lugar cuando se almacena la orden como línea de programas.

b) *Modo programa:*

Recordemos que un programa es un conjunto de instrucciones, que se ejecutan una tras otra según un orden preestablecido. Este orden viene dado por el número de línea que es condición suficiente para que el ordenador sepa que la orden que se escriba a continuación no debe ser ejecutada inmediatamente sino almacenada junto con las demás para constituir el programa.

Veamos un ejemplo:

Escriba: **10 PRINT "YO SOY EL ZX81"** y pulse NEW LINE. Entonces observará que, en lugar de ejecutar la orden, ésta pasa a la parte superior de la pantalla, como una línea de programa, con número 10 y sólo se ejecutará cuando se lo ordenemos con RUN. Al hacerlo, obtendrá la siguiente imagen:

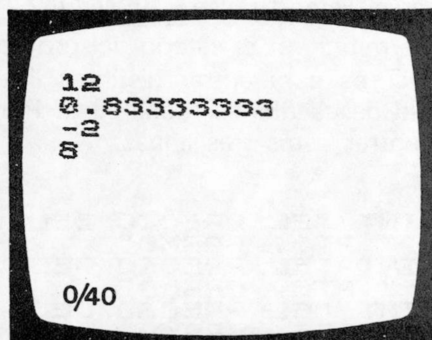


Como ya vimos, el 0 nos indica «que todo va bien» y el 10 nos dice que la última instrucción ejecutada fue de la línea 10.

Edición (modificación de líneas de programa):
Escriba un programa más largo:

```
10 PRINT 5+7
20 PRINT 5/6
30 PRINT 2*(4-5)
40 PRINT 4*2
```

Para hacerlo, escriba los números de líneas, las instrucciones y al terminar cada una, pulse NEW LINE y al final obtendrá en la pantalla la siguiente imagen:

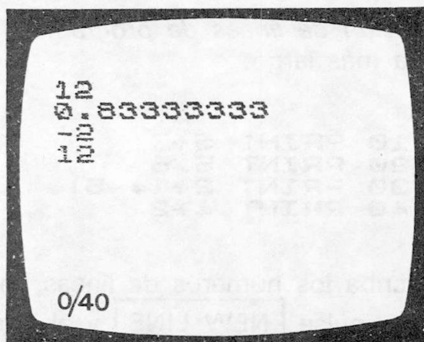


El ⊵ nos indica la línea actual, que en este caso es PRINT 4 * 2. Supongamos que queremos modificar esta línea poniendo PRINT 4 * 3. Para ello debemos bajarla con EDIT (SHIFT 1) y una vez abajo colocar un 3 en lugar del 2. Una vez hecho esto pulse NEW LINE y la nueva línea sustituirá a la antigua.

Para modificar cualquier otra línea, sitúe el indicador de línea actual (⊵) en la línea que quiere modificar. Para ello utilice ⇑ y ⇓ (SHIFT 7 y SHIFT 6 respectivamente) y proceda del modo descrito anteriormente.

Si introduce una línea con el mismo número de otra que ya está en programa, la nueva es entrada y la anterior se pierde completamente.

Ejecute el programa con RUN y obtendrá:



Para volver a ver el listado pulse NEW LINE o LIST . Esta última instrucción permitirá ver el estado del programa.

Otro uso de EDIT es para entrar distintas líneas que sean parecidas sin necesidad de escribirlas todas ellas. Por ejemplo, supongamos que quiere entrar estas tres líneas:

```
10 PRINT "EL PRECIO DEL ORO ES
";1600;" PTS./GRAMO"
 20 PRINT "EL PRECIO DEL ORO ES
";1800;" PTS./GRAMO"
 30 PRINT "EL PRECIO DEL ORO ES
";2000;" PTS./GRAMO"
```

Para ello, introduzca la primera línea tal como se ha explicado pero la segunda y tercera puede escribirlas pulsando EDIT y modificando el número de líneas y el precio según corresponde, con lo que se habrá ahorrado escribir lo mismo tres veces.

También puede usar EDIT , para borrar completamente la línea que se está escribiendo, en lugar de borrarla carácter a carácter haciendo EDIT , NEW LINE ya que al pulsar EDIT se produce la edición de la línea actual y por tanto la desaparición de la que se está escribiendo. Al pulsar NEW LINE , la línea actual vuelve a su sitio sin ser cambiada, con lo que el resultado final es que se ha borrado lo que se está escribiendo.

Ejercite todo esto escribiendo programas y modificándolos con EDIT .

CAPITULO 4

EMPEZANDO A PROGRAMAR

Variables y constantes

Los datos almacenados en la memoria del ZX81 son etiquetados con nombres simbólicos que indican al ordenador la posición de memoria en que están almacenados. Por ejemplo, la instrucción LET A = 2, almacenará en memoria el número 2 con el nombre A. Si nosotros decimos PRINT A, escribirá 2. Como que estos datos pueden ser cambiados mediante otra instrucción LET, por ejemplo LET A = 3, hace que reciban el nombre de variables. A la acción de etiquetar un..dato con un nombre N se le llama *asignación.*

Tipos de variables

El ZX81 considera dos tipos de variables: las *numéricas* (que contienen números) y las *cadenas o alfanuméricas* (que contienen caracteres). Los nombres simbólicos que se les pueden dar dependen del tipo de variable.

Nombres de las variables numéricas

Pueden ser de cualquier longitud (conteniendo letras, números o espacios) pero empezando forzosamente por una letra (no se admiten caracteres en negativo). Los espacios no tienen ningún efecto.

Nombres correctos:

PROD1
SOY EL ZX81
A
S1234

Nombres incorrectos:

A * 38 (* no es ni una letra ni un número)

1PROD (no empieza por una letra)

S1 H D (Un carácter en inverso tampoco es una letra para el ZX81)

Nombres de las variables alfanuméricas

Constan de una letra seguida del carácter $. Por ejemplo, A$, B$.

Constantes

Distinguiremos tres tipos de constantes: los enteros, los reales, de los que ya hemos hablado, y los exponenciales.

La representación exponencial, no es más que una notación científica en la que la potencia de 10 va precedida por la letra E.

Así, el número: $0{,}6154 \times 10^{-28}$ se representa como 0,6154E-28 en el ZX81.

Esta notación es muy útil para representar números muy grandes o muy pequeños. El número mayor capaz de ser almacenado por el ZX81 es aproximadamente el 1×10^{38} y el número positivo más pequeño es el 3×10^{-39}, aproximadamente. Estos números son retenidos con una precisión de 9 ó 10 dígitos y la razón de ello la veremos cuando estudiemos la forma de almacenar las variables.

Expresiones

Una expresión es un conjunto cualquiera de variables o de constantes, relacionadas entre sí mediante operaciones aritméticas o funciones. Más adelante veremos también ejemplos de expresiones que consisten en variables o constantes, relacionadas mediante operadores lógicos.

Ejemplos de expresiones son: −A**(B−C)
−3 + 4*A−COS B
−4
−A etc...

Para hacer más sencilla la programación con el ZX81 desarrollaremos antes la idea de lo que es un programa y pondremos algún ejemplo.

Mucha gente tiene la idea de que un ordenador es algo así como un genio que lo sabe hacer todo. Nada más falso. Un ordenador no hace absolutamente nada que no se le haya mandado previamente y esto resulta muy importante recordarlo pues es una idea básica que debe tenerse en cuenta a la hora de empezar a programar.

Por ejemplo, no podemos decirle a un ordenador que haga unas operaciones con unas variables si no conoce el valor que tienen. Es decir, si todavía no han sido definidas.

Hemos dicho ya muchas veces que el programa es un conjunto de instrucciones. Vamos ahora a hablar de ellas.

Las instrucciones, también llamadas sentencias, deben tener un orden de ejecución. Este orden puede ser, por ejemplo, el mismo en el que se han dado al ordenador, pero lo más normal y conveniente es numerarlas. Se ejecutarán entonces por el orden natural de los números, independientemente del orden en que han sido introducidas.

Las instrucciones, como ya sabemos, se escriben en lenguaje de alto nivel, el cual tiene un número limitado pero suficiente de palabras que, combinándolas adecuadamente, permiten resolver, con mayor o menor dificultad, cualquier problema de programación que se nos presente.

Proceso a seguir para programar

Existen muchos métodos distintos que enseñan cómo programar adecuadamente, pero todos coinciden en que hay que seguir el siguiente proceso:

1.° Tener idea clara del problema que se quiere resolver y esto no es siempre tan fácil como parece a simple vista.

2.° Encontrar el método adecuado que nos resuelva el problema planteado.

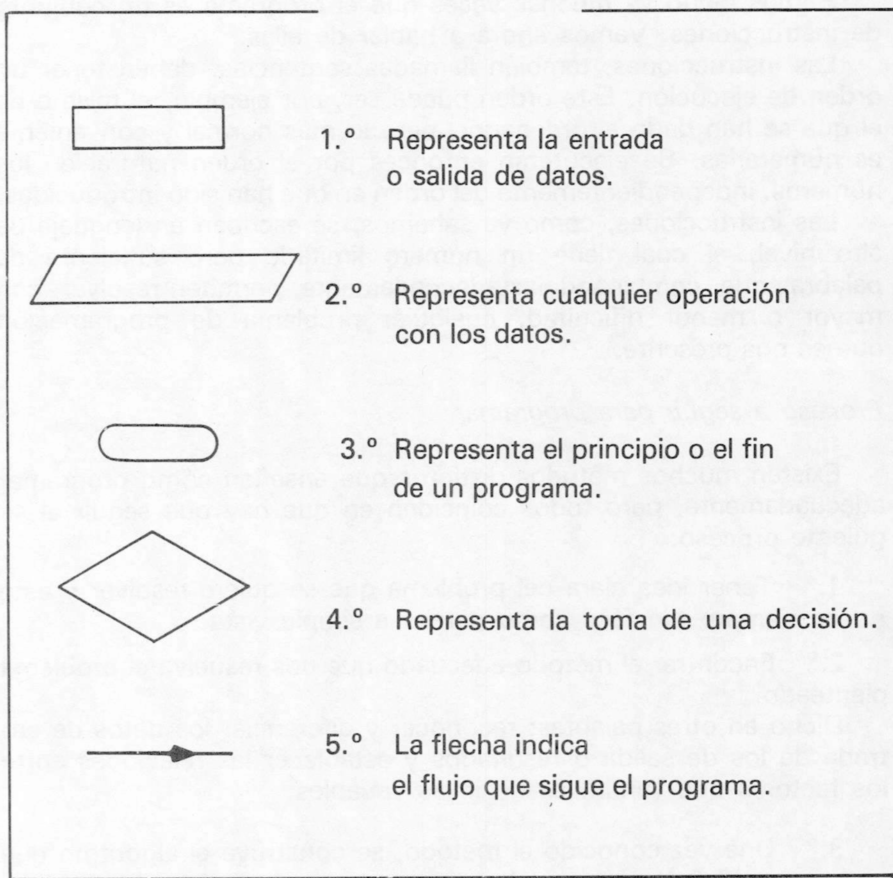
Dicho en otras palabras: reconocer y diferenciar los datos de entrada de los de salida o resultados y establecer las relaciones entre los factores que varían, es decir las variables.

3.° Una vez conocido el método, se construye el algoritmo que es la especificación exacta de las operaciones que hay que realizar con los datos de entrada para obtener los resultados.

4.° Una vez conocido el algoritmo, viene lo más fácil: simplemente, hay que traducir éste al lenguaje de alto nivel en el que queremos hacer el programa.

Los diagramas de flujo

Una forma muy corriente de planificar un programa es utilizar los llamados diagramas de flujo, en los que unas figuras geométricas representan los distintos procesos que se ejecutan. En la mayoría de los casos se utilizan los siguientes símbolos:



Veamos dos ejemplos de diagramas de flujo:

1.º Diagrama de flujo de un programa para atravesar una carretera (fig. 11).

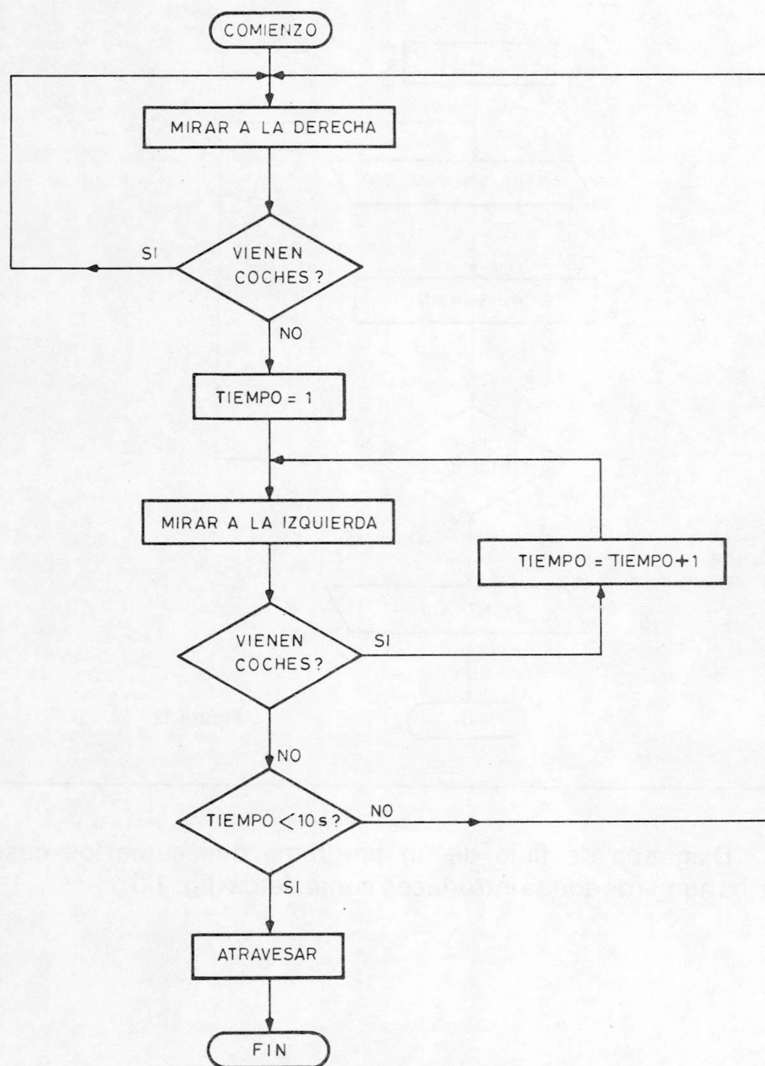

COMIENZO
MIRAR A LA DERECHA
SI
VIENEN COCHES ?
NO
TIEMPO = 1
MIRAR A LA IZQUIERDA
TIEMPO = TIEMPO+1
VIENEN COCHES?
SI
NO
TIEMPO < 10 s ?
NO
SI
ATRAVESAR
FIN

Figura 11

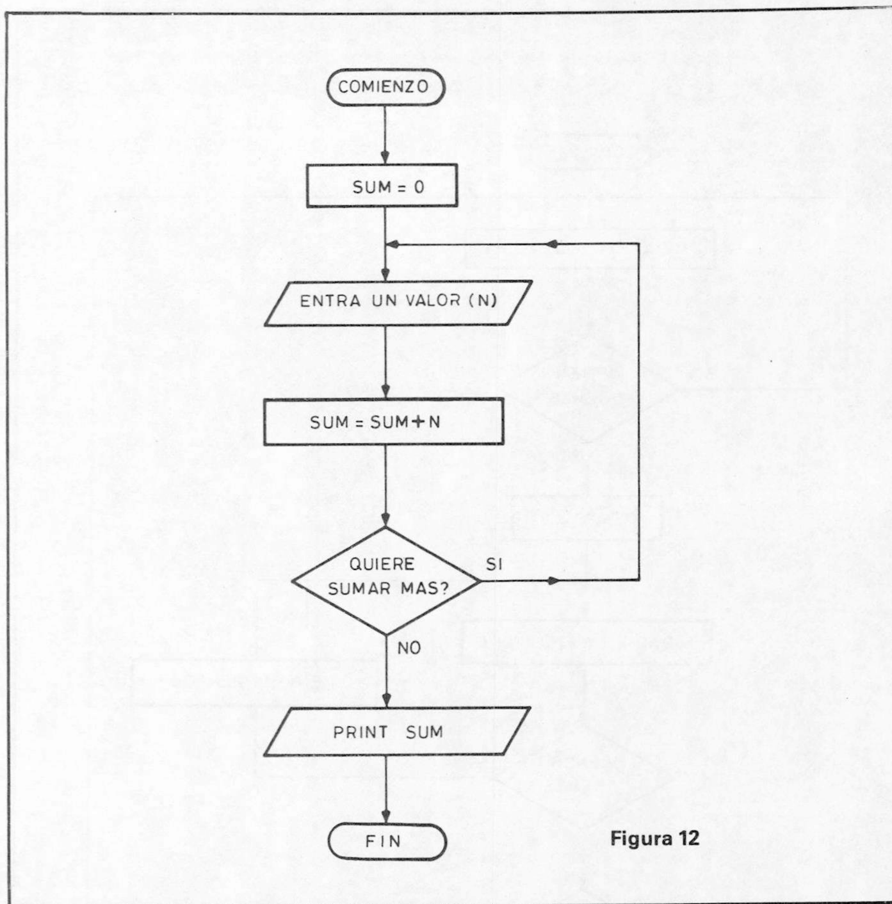


Figura 12

2.° Diagrama de flujo de un programa que suma los cuadrados de los números que se introducen como datos (fig. 12).

CAPITULO 5

INSTRUCCIONES PRINCIPALES

En este capítulo se ofrecen las instrucciones principales del BASIC del ZX81. Con ejemplos de utilización.

Instrucción LET

La instrucción de LET tiene el siguiente formato:

LET N = expresión.

Donde N es cualquier nombre de variable. Si el nombre es nombre de cadena (variable alfanumérica), la expresión debe ser también de cadena, y si el nombre es numérico, la expresión debe ser numérica. LET N = expresión actúa asignando a la variable cuyo nombre es N el valor de la expresión.

Ejemplo:

LET NUM = 100, asigna el valor cien a la variable numérica NUM.

LET PROD = NUM × 3 + 4, asigna a PROD el valor de NUM multiplicado por tres y añadiendo cuatro a todo ello.

LET A$ = "PEDRO" asigna a la variable A$ la cadena de caracteres "PEDRO".

Las constantes alfanuméricas siempre se enmarcan entre comillas.

Instrucción INPUT

La instrucción INPUT sólo se puede usar en programa, no como una orden. Su formato es:

Número de línea INPUT N

Donde N es cualquier nombre de variable y actúa del siguiente modo:

Cuando el programa llega a la línea que contiene INPUT, se para y se espera a que se introduzca un dato (ya sea una cadena o numérico, lo cual depende del nombre de la variable).

Esto se observa porque aparece en la parte inferior izquierda de la pantalla el cursor L. Una vez introducido el dato, la instrucción INPUT actúa como LET N = dato introducido.

Si el nombre de variable es de cadena, el cursor L aparece con dos comillas y se puede introducir cualquier carácter del teclado. Si el nombre de la variable es numérico, sólo se pueden introducir datos numéricos ya que, en caso contrario, aparecería error.

Instrucción PRINT

La instrucción PRINT se usa para obtener información por la pantalla. Su formato es:

PRINT 1.° campo a imprimir; PRINT 2.° campo a imprimir...

Puede tener distintas variaciones que explicaremos con ejemplos:

1.° PRINT A: el valor de la variable A es escrito en la pantalla en caso de que exista. En caso contrario, aparecerá error 2 (Consúltese Capítulo 20: «Códigos de Error»).

```
10 INPUT A
20 PRINT A
```

2.° PRINT A$: el valor de la variable de cadena A$ es escrito en la pantalla.

```
10 INPUT A$
20 PRINT A$
```

3.° PRINT «cadena»: las comillas en la instrucción PRINT hacen que en la pantalla se imprima exactamente lo que hay entre ellas. En este caso aparecerá la palabra «cadena».

Con la instrucción PRINT es posible escribir varias cosas a la vez, pudiendo ir separadas por coma o punto y coma. El punto y coma hace que el siguiente dato a imprimir lo haga yuxtapuesto al anterior, mientras que la coma hace que el siguiente dato a imprimir sea escrito o bien en la columna 16, si el anterior no la ocupaba ya o, en caso contrario, se imprima en la columna 0 de la siguiente línea.

Una sentencia PRINT sin ningún campo a imprimir, deja una línea en blanco.

Escriba el siguiente programa que le ayudará a ver cómo actúan las distintas posibilidades que tiene la instrucción PRINT:

```
10 INPUT A
20 INPUT A$
30 PRINT A,A,A,A
40 PRINT A$;A$;A$;A$
50 PRINT A$;" ";A$;" ",,,,"EST
E PROGRAMA NO HACE ABSOLUTAMENTE
NADA"
```

Ejecútelo con RUN e intente comprender el por qué de la impresión de los dos tipos de datos que se piden, relacionándolo con lo que se acaba de ver.

Veamos un ejemplo que utilice estas instrucciones que hemos visto; LET, PRINT, INPUT.

```
10 PRINT "CUAL ES SU NOMBRE?"
20 INPUT A$
30 PRINT "QUE EDAD TIENE?"
40 INPUT A
50 LET E=A-3
60 PRINT "USTED ES ";A$;" Y TI
ENE ";E;" ANOS MAS QUE EL ZX81"
```

Instrucción LIST

La instrucción LIST tiene el formato:

LIST N; donde N es número de línea. Actúa listando el programa desde la línea N y situando en dicha línea el indicador de línea actual. Es pues muy útil para situar el cursor de línea actual donde nos interese sin necesidad de usar ⇑ y ⇓ . LIST actúa como LIST 0.

Instrucción CLS

La instrucción CLS borra la pantalla. Si la pantalla está llena y se quiere imprimir algo en ella, debe ser borrada pues, en caso contrario, aparecería error 5 (Consúltese Capítulo 20).

Instrucción NEW

La instrucción NEW borra todo lo almacenado en el ZX81, excepto las rutinas que se encuentran por encima del valor indicado en la variable RAMTOP (Capítulo 15). De momento, para nosotros, hacer NEW tiene el mismo valor que desconectar y volver a conectar el ZX81.

Instrucción CLEAR

La instrucción CLEAR borra únicamente las variables, no el programa. Ejemplo:

```
10 INPUT A
20 PRINT A
30 CLEAR
40 PRINT A
```

Observe cómo la primera vez se imprime el valor de la variable que usted introduce; en cambio, después de CLEAR se produce error 2, ya que la variable no está definida. En cambio el programa continúa en su lugar.

Instrucción RUN

La instrucción RUN no se usa normalmente como línea de programa salvo en muy determinadas ocasiones. Su formato es:

RUN N, donde N es número de línea y actúa ejecutando el programa a partir de la línea N, y —¡atención a esto!— previamente borra todas las variables.

Si se quiere ejecutar un programa sin que se borren las variables, hay que utilizar la instrucción GOTO.

Instrucción GOTO

El formato de la instrucción GOTO es:

GOTO expresión y actúa saltando a la línea cuyo número es el resultado de la expresión redondeado al entero más próximo. La diferencia con RUN es que GOTO no borra las variables. Por lo tanto, RUN es equivalente a CLEAR y GOTO N.

Veamos un programa de ejemplo en el que se utilicen estas instrucciones:

```
10 PRINT "PRIMER NUMERO:";
20 INPUT A
25 PRINT A
30 PRINT "SEGUNDO NUMERO:";
40 INPUT B
45 PRINT B
50 PRINT "TERCER NUMERO:";
60 INPUT C
70 PRINT C
80 LET SUM=A+B+C
90 LET PROD=A*B*C
100 PRINT ,,,,A;"+";B;"+";C;"="
;SUM
110 PRINT ,,A;"*";B;"*";C;"=";P
ROD
120 CLS
130 GOTO 10
```

Este programa pide tres números y calcula su suma y su producto, que se almacenan en las variables SUM y PROD, respectivamente. Observe que si se suprime la línea 120 (CLS), llega un momento en el que se llena la pantalla y se detiene el programa. Más adelante veremos cómo se subsana esto utilizando CONT.

Instrucción STOP

La instrucción **STOP** sirve para detener un programa y se puede utilizar de dos maneras:

1.° Como línea de programa y entonces el programa se detiene con el informe 9/N, donde 9 indica que se ha parado debido a un STOP y N es el número de línea en que éste se encuentra.

2.° STOP en INPUT, si se quiere detener un programa cuando éste está pidiendo un dato mediante la instrucción INPUT. Si en lugar de entrar el dato, se pulsa STOP ([shift] [A]), éste se detiene. Si el dato que pide es una cadena, hay que borrar previamente las primeras comillas, pues si no, se asignaría a la cadena la palabra STOP.

Instrucciones BREAK

La instrucción **BREAK** no se utiliza como línea de programa y sirve para detener un programa cuando está funcionando y proporciona el informe D/N (Véase Capítulo 20).

Instrucción PAUSE

El formato de la instrucción PAUSE es:

PAUSE N y actúa deteniendo el programa durante N/50 segundos aproximadamente, si N es menor que 32.767. Si hacemos PAUSE con un valor mayor de 32.767, el tiempo no se contabiliza y el programa queda detenido. En cualquiera de estos dos casos, puede interrumpirse la pausa simplemente pulsando una tecla. En ejemplos posteriores se verán distintas maneras de utilizarse PAUSE.

Instrucción CONT

La instrucción **CONT** se utiliza para reemprender la ejecución de un programa detenido mediante una sentencia STOP o mediante BREAK. CONT borra la pantalla y reemprende la ejecucíon del programa a partir de la última línea, pero si quedó detenido por una línea que contuvo STOP, se empieza a ejecutar desde la línea siguiente.

Para ver cómo funciona, se suprimirá la línea 120 del programa anterior y, al detenerse con error 5, se pulsará CONT y el programa continuará.

Instrucción REM

La palabra REM proviene del inglés REMARK, que significa comentario y ésta es exactamente la función de la instrucción REM: tener la posibilidad de incluir comentarios en el listado mismo del programa. Cuando el ZX81 se encuentra con una sentencia REM, simplemente se olvida de ella; es decir, todo lo que hay en la línea no es procesado por el ordenador. Su formato es:

10 REM comentario

Más adelante veremos que el comentario incluido en la sentencia REM puede ser un programa en código máquina situado en este lugar tan peculiar para que no sea sobreescrito por ninguna instrucción en BASIC.

Instrucción SCROLL

La instrucción **SCROLL** elimina la línea superior de la pantalla y la próxima línea a imprimir lo hará en la parte baja. Esto es muy útil cuando se tienen que presentar por pantalla más líneas de las que caben, evitando, mediante SCROLL, la detención del programa y el tener que reanudarlo con CONT.

Para ver cómo funciona todo esto, escriba el siguiente programa y observe la diferencia si se suprime la línea que contiene SCROLL.

```
10 INPUT A
20 SCROLL
30 PRINT A
40 GOTO 10
```

Este programa es una simulación de una entrada de datos que no se detiene nunca debido al llenado de la pantalla, gracias a la instrucción SCROLL.

GOSUB y RETURN: Subrutinas

Muchas veces nos encontramos con que en un mismo programa, se tiene que realizar varias veces un mismo proceso. Por ejemplo, la impresión de resultados o la entrada de datos. Para evitar el tener que escribir varias veces lo mismo está la combinación de estas dos instrucciones: GOSUB y RETURN.

La instrucción GOSUB actúa exactamente igual que GOTO, con la única diferencia de que en un área de la memoria se almacena el número de línea del que se ha partido y cuando el programa se encuentra con una instrucción RETURN, vuelve a la línea siguiente a la que se ha almacenado.

El trozo de programa comprendido entre la línea a la que salta GOSUB y la instrucción RETURN, es llamada subrutina. Veamos ahora un ejemplo muy sencillo pero que da una idea de la utilidad de las subrutinas.

```
  20 LET I=1
  25 SCROLL
  30 PRINT "NO ESTOY EN LA SUBRU
TINA"
  35 SCROLL
  40 GOSUB 5000
  50 LET I=I+1
  60 GOTO 25
5000 PRINT "PASO POR LA SUBRUTIN
A POR ";I;" VEZ"
5010 RETURN
```

Las subrutinas no son imprescindibles para programar, puesto que se pueden sustituir por GOTOs convenientemente situados, pero ayudan mucho a la comprensión de la estructura del programa que aparece mucho más clara, dividida en módulos funcionales e independientes por sí mismos.

Cuando el programa pasa por una instrucción GOSUB, almacena la dirección, a la cual tiene que volver, en un compartimento llamado pila de GOSUB ya que se puede imaginar que las direcciones de retorno se almacenan unas encima de otras y se retiran (cuando acontece un RETURN) en el orden inverso; es decir, primero se coge la última dirección depositada, después la penúltima, etc.... Es como si estuvieran amontonadas de manera que, para regresar a una dirección, tuviera que haberlo hecho primero a todas las que están encima.

CAPITULO 6

OPERACIONES Y FUNCIONES

Operaciones

El ZX81 distingue entre dos tipos de operaciones: las operaciones aritméticas y las operaciones lógicas (que serán tratadas en otro Capítulo).

Las operaciones aritméticas son las ya conocidas por todos: suma +, resta —, multiplicación *, división /, potenciación ** y el llamado menos de signo que es el que indica, en la expresión a la que afecta, que tiene el valor cambiado de signo.

Por ejemplo: —3 cambia el signo al número 3.

—(3—7) en este caso, el menos de signo, que es el que se encuentra delante del paréntesis, no el que está dentro, hace que el valor de esta expresión sea + 4 en lugar de —4.

La distinción que se hace entre estos dos tipos de símbolo, es debida únicamente a la prioridad a la hora de ejecutarse las operaciones.

Prioridades

Cada operador tiene asignado un número que indica la prioridad que tiene respecto a los demás. Estos valores de prioridad se muestran en la figura 13.

Ante una expresión, el ZX81 actúa de la siguiente manera para encontrar su valor: primero busca todos los valores que tienen la prioridad más alta y evalúa las operaciones de izquierda a derecha. Des-

OPERADOR	PRIORIDAD
★ ★	10
–	9
★	8
/	8
+	6
–	6

Figura 13

pués busca los de la segunda prioridad más alta, actuando de igual forma y así sucesivamente.

```
7*2**3/4+3
7 * 8 / 4+3
56 / 4+3
14+3
17
```

En caso de que se quieran variar las prioridades, se pueden usar los paréntesis (SHIFT 1, SHIFT 0). Como se ve en este ejemplo:

```
7*2*x3/(4+3)
7*2x*3/  7
7*8/7
56 / 7
8
```

Funciones

Podemos definir una función como una regla que sustituye un número, conocido como argumento, por otro que es el resultado.

Ejemplo: Cos 60. La función coseno, aplicada al ángulo de 60, nos da el valor de 0,5. Aquí 60 es el argumento y 0,5 es el resultado.

En el ZX81, las funciones se encuentran debajo de las teclas y para que se impriman hay que transformar el cursor a modo F. Esto se consigue pulsando SHIFT F— NL y la próxima tecla que se pulse provocará la impresión de la función que le corresponde.

Podemos clasificar las funciones del ZX81 en tres grupos:

1.° *Funciones matemáticas:* Son SIN, COS, TAN, INT, ARCSIN, ARCCOS, ARCTAN, SGN, ABS, SQR, LN, EXP y PI.

2.° *Funciones aplicables a cadenas:* Son VAL, LEN, STR, CHR$, CODE. Estas se estudiarán en el Capítulo dedicado a cadenas.

3.° *Funciones especiales:* Son RND, PEEK, USR, INKEY$, NOT.

Las otras palabras que están debajo de las teclas: TAB, AT, no son realmente funciones pero hay que pulsar igualmente SHIFT F—NL para conseguirlas.

Funciones matemáticas

El ZX81 tiene seis funciones trigonométricas: SIN, COS, TAN, ARCSIN, ARCCOS, ARCTAN. Estas funciones tratan ángulos y veremos a continuación cómo actúan.

En un triángulo rectángulo, se define el seno de un ángulo, que no sea el de 90, como la razón entre la hipotenusa y el cateto opuesto, mientras que el coseno es la razón entre la hipotenusa y el cateto adyacente. La tangente es la razón entre cateto opuesto y cateto adyacente (figura 14).

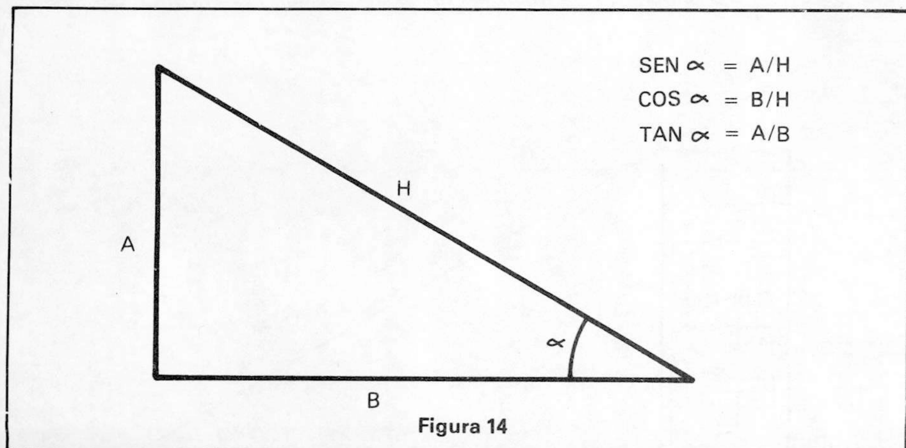


Figura 14

El argumento de SIN, COS y TAN, es un ángulo que debe ser expresado en radianes. Como sabemos que 2 π radianes = 360 (π = PI), a un ángulo en grados hay que multiplicarlo por PI y dividirlo por 180 para pasarlo a radianes.

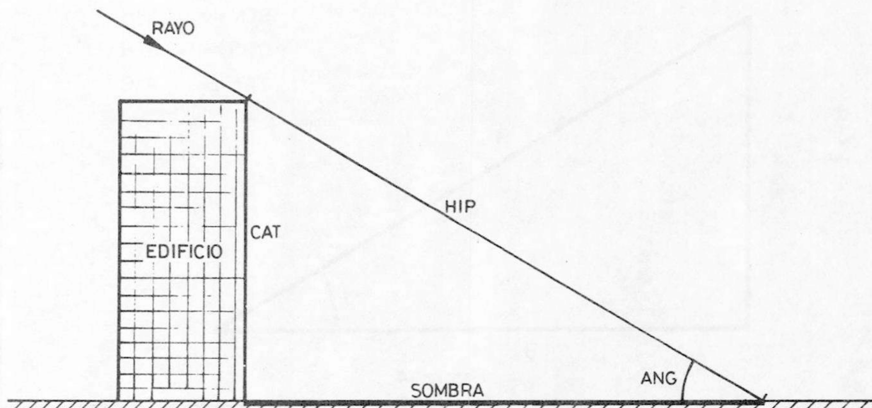
El seno y el coseno siempre toman valores entre —1 y 1, mientras que la tangente toma valores entre — infinito y + infinito.

Las funciones inversas de estas tres son ARCSIN que, aplicada a un número entre —1 y 1, nos da el valor del ángulo cuyo seno es dicho número. Lo mismo ocurre con ARCCOS y ARCTAN.

PI es una función sin argumento y proporciona la razón entre el perímetro y el diámetro de una circunferencia. PI = 3,1415927.

```
  10 PRINT "          INTRODUCCION DE
DATOS"
  20 PRINT ,,"ALTURA DEL EDIFICI
O ";
  30 INPUT CAT
  40 PRINT CAT
  50 PRINT ,,"ANGULO DEL SOL:";
  60 INPUT ANG
  70 PRINT ANG
  72 REM PASO A RADIANES
  75 LET ANG=ANG*PI/180
  78 REM CALCULO DE LA SOMBRA
  80 LET HYP=CAT/SIN ANG
  90 LET SOMBRA=HYP*COS ANG
  95 PRINT ,,,,"           RESULT
ADOS"
  100 PRINT ,,"LA SOMBRA PROYECTA
DA ES ";SOMBRA
```

Este programa calcula la longitud de la sombra proyectada por un edificio a partir de la altura de éste y del ángulo que forman los rayos del sol con la horizontal del suelo.



El edificio, el rayo y la sombra forman un triángulo rectángulo, lo cual permite calcular la hipotenusa y luego el cateto adyacente que es la sombra.

El ángulo debe ser introducido en grados pues el programa realiza la conversión a radianes en la línea 75.

Si se da como dato un ángulo muy pequeño, se puede producir un error 6 en la línea 80, debido a que resulta un valor muy grande.

Función ABS

La función ABS aplicada a un número X, nos da X si X >0 o si X = 0 y nos da —X si X < 0, es decir, el máximo entre X y —X.

Función SQR

La función SQR aplicada a un número X, calcula la raíz cuadrada ($\sqrt{X}$) si $X \geqslant 0$ y da error si $X < 0$.

Función EXP

La función EXP aplicada a un número X da como resultado e** X donde e = 2,7182818...

Función LN

La función LN calcula el logaritmo en base del argumento. Si éste es menor o igual que 0 da error. La función LN es la inversa de la exponencial. Por ejemplo, si se intenta hacer EXP LN 3, esto debe devolver el número 3. Si no lo hace exactamente es debido a que no puede retener las cifras de los cálculos.

La combinación de EXP y LN sirve para calcular cualquier potencia o raíz de un número, siempre que se tengan en cuenta las siguientes reglas:

*LN A * LN B = LN (A + B)*
LN A / LN B = LN (A—B)
*LN (A**N) = N + LN A*
LN (SQR A) = 0.5 LN A
EXP (A + B) = EXP A + EXP B
EXP (A — B) = EXP A / EXP B
*(EXP A)**B = EXP (A*B)*

DE ESTE MODO PODEMOS VER QUE

$$\sqrt[15]{54} \; EXP[(LN 54)/15]$$

Función INT

La función INT aplicada a un número X da como resultado el número entero inmediatamente inferior o igual que X.

INT 4 = 4 / INT 3,999 = 3 / INT −1,5 = −2

La función INT tiene mucha importancia pues sirve para calcular los restos de las divisiones.

Así el resto de la división A |B es A − INT (A/B)* B. Por ejemplo:

8 | 3
2 2 si hacemos

8 − INT (8/3)* 3 = 8− 2*3 = 2

Más adelante cuando se haya visto la programación en bucles y la sentencia condicional, contemplaremos un programa en el que, utilizando la función INT, convierte un número de una base a otra.

Si en lugar de obtener el entero inferior se quiere redondear el número al entero más próximo, se le debe sumar a éste 0,5 antes de aplicarle la función INT.

Otras funciones:

Función RND

La función RND carece de argumento y es lo que se llama un generador de números aleatorios. Si se prueba varias veces PRINT RND se observará que cada vez le sale un número distinto entre 0 y 1, aunque sin alcanzar nunca el valor 1. La manera de obtenerlos es complicada y no se explica aquí pues presupone conocimientos matemáticos elevados. Si lo que se quiere es obtener, por ejemplo, un número entero entre 1 y 6, deberá hacerse PRINT INT 6* RND + 1.

Como estos números son aleatorios pueden simular, por ejemplo, la tirada de un dado.

La función RND es muy útil para juegos y se verán algunos ejemplos en el Capítulo dedicado a gráficos.

La sentencia RAND controla la aleatoriedad de RND y actúa de la siguiente manera: si nosotros antes de hacer RND hacemos RAND × (1 ⩽ X ⩽ 872), el próximo valor de RND será (75*[X + 1] −1)/65536. En realidad lo que hace RND es seguir una secuencia de 65.536 números que parecen aleatorios porque están desordenados y lo que hace RAND es empezar la secuencia de acuerdo con un número determinado que ya ha quedado explicado cómo se calcula.

CAPITULO 7

EL JUEGO DE CARACTERES. — LAS CADENAS

El ZX81 dispone de una cierta facilidad para tratar textos. Esto se consigue mediante el uso de cadenas.

Una *cadena* es un carácter, ya sea numérico, gráfico, función, instrucción, etc., enmarcado entre comillas.

Ejemplo de cadenas son: "ZX81", "GOTO", "GOTO", etc. Se preguntará el lector qué diferencia hay entre los dos últimos ejemplos. Pues bien, a simple vista no hay ninguna pero nosotros podemos escribir GOTO de dos formas distintas: bien deletreando cada letra G, O, T, O o utilizando un ingenioso truco para «engañar» a la máquina.

El truco es el siguiente:

1.º Se abren comillas.

2.º El cursor se encuentra en modo L, con lo que, al pulsar la tecla G, simplemente nos aparecerá una G pero si nosotros pulsamos THEN y luego G, obtendremos THEN GOTO.

3.º Borramos THEN.

4.º Cerramos las comillas de cadena.

La diferencia entre los dos es muy simple. La primera consta de cuatro caracteres mientras que la segunda consta únicamente de uno. No hay diferencia en cuanto a la impresión que aparece en pantalla pero podemos observarla cuando sepamos cómo obtener la longitud de una cadena o bien los códigos de los caracteres que la componen.

Las cadenas pueden contener espacios y éstos tienen el mismo valor que los demás caracteres. Escriba:

```
10 LET A$ "MI CASA"
20 PRINT A$
```

Si dentro de una cadena se quieren introducir comillas, se debe hacer con las de la tecla Q que son presentadas como dobles comillas pero que, a la hora de imprimirse en una secuencia PRINT, se imprimen como simples. Esto es debido a que la inclusión dentro de una cadena de las comillas «simples», provocaría que el ZX81 las considerara como el final de la cadena pues el símbolo '', es el que marca el principio y el fin de la cadena. Escriba:

PRINT ''HOLA SOY'' ''ZX81'' '' ''

Las cadenas también se pueden asignar a variables pero ya vimos que éstas deben tener un nombre especial que consiste en una sola letra seguida del carácter $.

Operaciones con cadenas

1.° *Concatenación:*

La concatenación de dos cadenas consiste en añadir una al final de la otra.

Ejemplo: Escriba PRINT ''ME LLAMO'' + ''LUIS'' o bien

```
10 LET A$ = "ME LLAMO"
20 LET B$ = "LUIS"
30 LET C$ = "ANTONIO"
40 PRINT A$ + B$, A$ + C$
```

Esta operación se ejecuta con el símbolo +, el mismo que para sumar.

2.° Ordenación

Las cadenas se pueden ordenar alfabéticamente, utilizando los símbolos < =, > =, <, >, =, < >. Para ver cómo actúan, consúltese el Capítulo referido a operadores lógicos.

3.° Segmentación

La segmentación es una operación que nos permite obtener trozos de una cadena dada. Su forma general es la siguiente: «cadena» (N TO M). La cadena puede ser explícita, por ejemplo «MESA», o bien una variable de cadena por ejemplo A$.

Esta operación actúa de la siguiente forma: obtiene de la cadena que le precede, los caracteres que van desde el N-esimo al M-esimo,

ambos inclusive. Si N es menor que 1 ó M mayor que la longitud de la cadena se producirá un error 3. Escríbase:

```
PRINT "ABCDE" (3 TO 5), o bien
10 LET A$ = "ABCDE"
20 PRINT A$ (3 TO 5)
30 PRINT A$ (1 TO 3)
```

La segmentación tiene unos valores por defecto que son los siguientes:

A$(N TO): obtiene los caracteres de la cadena A$ desde N hasta el final.

A$(TO N): obtiene los caracteres de la cadena A$ desde el principio hasta N.

A$(N): obtiene únicamente el carácter N-esimo.

A$(TO): es lo mismo que A$.

```
  10 PRINT "ESCRIBA UNA PALABRA
DE DIEZ LETRAS"
  20 INPUT A$
  30 PRINT "LAS TRES PRIMERAS LE
TRAS SON: """;A$( TO 3);""""
  40 PRINT "LAS TRES ULTIMAS LET
RAS SON: """;A$(8 TO );""""
  50 PRINT "LA SEXTA LETRA ES: "
"";A$(6);""""
```

Se puede considerar que la segmentación tiene prioridad 12.

Funciones de cadenas

Cada carácter del ZX81 tiene un código que va desde 0 a 255. Con esto vemos que un carácter puede ser registrado en un byte.

Hay dos funciones que nos permiten saber qué carácter corresponde a un número o bien qué código corresponde a un carácter.

Función CHR$

La función CHR$ no se aplica a cadenas sino que aplicada a un número entero mayor que 0 y menor que 255, obtiene el carácter

cuyo código es dicho número. Este programa que vemos a continuación escribe todos los caracteres del ZX81:

```
10 LET I=0
20 PRINT I,CHR$ I
30 LET I=I+1
40 GOTO 20
```

Función CODE

La función CODE, aplicada a una cadena, da como resultado el código del primer carácter de esa cadena. CODE puede servir para ver la diferencia existente entre los dos tipos de «GOTO» que se comentó al principio del Capítulo. Escríbase:

PRINT CODE "GOTO"(letra a letra)
y PRINT CODE "GOTO" (instrucción)

Obsérvese cómo en el primer caso se imprime el código de la letra G que es 44 (como se observa en el programa anterior), mientras que en el segundo caso lo que imprime es el código de la instrucción GOTO es decir 236.

Función LEN

La función LEN calcula la longitud de una cadena. Actúa del siguiente modo: LEN A$ = número de caracteres A$. Hágase lo mismo que en el caso anterior y se obtendrá que la longitud de la primera cadena es 4, mientras que la de la segunda es 1.

Función STR$

Esta función es muy sutil. Se aplica a un número o a una variable numérica y obtiene lo que aparecería en la pantalla si la variable o el número fuera representado por una sentencia PRINT.

Supongamos que tenemos una variable A que contiene un cierto valor y que queremos operar con ella como si de una cadena se tratara. Por ejemplo, queremos saber cuántas cifras tiene A. No podemos aplicar la función LEN directamente a A ya que A no es ninguna

cadena. Para esto está la instrucción STR$; pues nos convierte la variable A en una cadena conteniendo las cifras del número A.

```
10 PRINT "ESCRIBA UN NUMERO"
20 INPUT A
30 LET A$=STR$ A
40 LET L=LEN A$
50 PRINT "EL NUMERO ";A;" TIEN
E ";L;" CIFRAS"
```

Función VAL

La función VAL calcula el valor de una cadena si se considera a ésta como una expresión.

VAL tiene algunas restricciones:

1.° Si la función VAL es parte de una expresión más larga, debe estar en primer lugar.

2.° Si la cadena a la que se aplica VAL, contiene variables, entonces VAL sólo puede aparecer en la primera coordenada de una sentencia PRINT, PLOT, UNPLOT.

Estas instrucciones se verán en el Capítulo dedicado a los gráficos.

Ejemplo:

Escriba lo siguiente sin número de línea:
*LET A$ = " A*X**2 + B*X + C" y seguidamente introduzca el siguiente programa:*

```
10 INPUT A
20 INPUT B
30 INPUT C
40 INPUT X
50 PRINT A$;"=";VAL A$
```

Esto permite calcular el valor de la expresión contenida en A$ sin necesidad de definirla cada vez, lo cual permite sustituir en algunos casos la carencia en el ZX81 de funciones definidas por el usuario, que sí existen en otros BASICS. No ejecute el programa con RUN, pues se borraría el valor asignado a A$.

Las cadenas también se pueden combinar para obtener expresiones

encadenadas, teniendo en cuenta las prioridades que rigen en estos casos.

Ejemplo:

```
10 LET A$ = "ABCDEFG"
20 LET B$ = "MNOPQRST"
30 PRINT A$ (TO 6) + "G" +"HIJKLM" (TO 5) + B$ + "UVXYZ"
```

Función INKEY$

La función INKEY$ carece de argumento y su resultado es siempre una cadena que contiene el carácter de la tecla que se pulsa en el momento en que el programa pasa por la instrucción. Este carácter es el que se obtiene cuando el cursor está en modo L, aunque si se pulsa SHIFT, al mismo tiempo, se obtiene el carácter que está en rojo.

Si usted utiliza INKEY$ como una orden: PRINT INKEY$, puede ocurrir que no obtenga nada, pero lo más normal es que se imprima un ? que es como se representa el NEW LINE. Todo depende de la velocidad con que se retire el dedo de la tecla NEW LINE.

Al escribir este sencillo programa, sorprenderá las posibilidades que puede ofrecer INKEY$, que se aprovechan al máximo en el Capítulo dedicado a los gráficos.

```
10 PRINT INKEY$;
20 GOTO 10
```

Hay que tener muy en cuenta que la función INKEY$ no se espera a que Vd. pulse una tecla y si no lo hace en el momento en que el programa pasa por la instrucción, su resultado es la cadena vacía. Para resolver esto disponemos de la instrucción PAUSE, que si se ejecuta con un valor mayor que 32.768 se mantiene hasta que una tecla es pulsada.

Este hecho es el que se utiliza en el siguiente programa en la línea 20.

```
10 PRINT "PULSE 1, 2 O 3"
20 PAUSE 35000
30 GOTO 100*VAL INKEY$
```

```
100 PRINT "HA PULSADO EL 1"
140 PAUSE 200
150 CLS
160 GOTO 10
200 PRINT "HA PULSADO EL 2"
240 PAUSE 200
250 CLS
260 GOTO 10
300 PRINT "HA PULSADO EL 3"
340 PAUSE 200
350 CLS
360 GOTO 10
```

Este programa es interesante porque toma decisiones distintas según la tecla que se pulse y, gracias a la línea 30, se dirige al lugar conveniente.

Otra manera de que INKEY$ se esperara sería poder decir:

10 SI INKEY$ = " " ENTONCES GOTO 10, lo que ya será posible en cuanto se haya leído el próximo Capítulo.

Como ejercicio es conveniente realizar un programa que, basado en el primero, actúe como una máquina de escribir. Para ello se tendrá que usar PAUSE.

CAPITULO 8

CONDICIONES Y OPERADORES LOGICOS

Los programas no tienen por qué ejecutarse forzosamente en forma lineal. Es decir, que existen instrucciones que permiten desviar el curso del programa en un momento determinado. Ejemplos de ellas son GOTO y GOSUB, que son instrucciones de desviación incondicionales ya que, cuando el programa llega a una de estas instrucciones, cambia su curso irremediablemente (salvo en situaciones triviales). Pero también existe una instrucción llamada *condicional* que permite actuar de diferente manera según se cumpla o no una condición determinada.

Veamos ahora cuál es la instrucción que nos permite desviar el curso del programa al cumplirse una condición. Esta instrucción es la combinación de dos palabras IF THEN. El formato es el siguiente:

IF (condición), THEN (sentencia) y actúa de la siguiente forma: si se cumple la condición, entonces ejecuta la sentencia; en cambio, si la condición no se cumple, salta a la línea siguiente.

Veamos esto más detalladamente. La condición, es algo susceptible de ser verificado, de manera que puede resultar verdadero o falso. Si resulta verdadero, se ejecuta la sentencia que puede ser, por ejemplo, una instrucción como GOTO, INPUT, LET.

Ejemplo: 10 IF A > 3 THEN GOTO 500

Cuando el programa pase por esta línea, si A tiene un valor mayor estrictamente que 3, entonces el programa saltará a la línea 500. En caso contrario, continuará con la línea siguiente.

Las condiciones más utilizadas son las que comparan números o cadenas y son:

>	Significa «es mayor que»
<	Significa «es menor que»
< =	Significa «es menor o igual que»
> =	Significa «mayor o igual que»
=	Significa «igual que»
< >	Significa «distinto que»

Ejemplo

Estas relaciones tienen prioridad 5 y usualmente se escriben <, >, ⩽, ⩾, =, ≠. Pero esto no es posible en el ZX81 y recuerde que para escribir > = y < =, se utiliza una sola tecla.

Estas relaciones se pueden combinar utilizando las operaciones lógicas AND, OR y NOT.

AND actúa del siguiente modo: Si las dos relaciones unidas por AND son verdad, entonces el resultado es verdad. En caso de que una de las dos o las dos sea falsa, entonces el resultado será falso.

OR: la operación lógica OR adquiere resultado verdadero siempre que una de las dos relaciones sea verdad. También si lo son ambas.

NOT: Sólo se aplica a una relación y es verdad cuando la relación es falsa y es falsa cuando la relación es verdadera.

Las relaciones <, >, < =, > =, =, < >, cuando se aplican a cadenas tienen el mismo significado sólo que el orden considerado es el orden alfabético. Más exactamente se ordenan por códigos, que coinciden con el orden alfabético cuando se trata de letras.

```
  10 LET M$=""
  15 PRINT "ESCRIBA UNA PALABRA"
  20 INPUT A$
  30 IF A$>=M$ THEN LET M$=A$
  40 PRINT "LA MAYOR HASTA EL MO
MENTO EN ORDEN ALFABETICO ES:",,
,,,,M$,,,,,,,
  50 PRINT "PULSE UNA TECLA PARA
 CONTINUAR"
  60 PAUSE 35000
  70 CLS
  80 GOTO 15
```

Este programa compara la cadena que se introduce con la mayor hasta el momento e imprime la mayor de las dos en cuanto a orden alfabético se refiere.

Si no comprende el por qué de la línea 10, suprímala y observe lo que ocurre.

Los operadores lógicos pueden constituir expresiones. Por ejemplo: A > 3 AND (B = 4 OR A < 7) y, por tanto, tienen unas prioridades que son NOT 4, AND 3 y OR 2.

Tabla general de prioridades

Ahora que ya conocemos las funciones y las operaciones, podemos escribir una tabla general de prioridades.

OPERADOR	PRIORIDAD
SUBINDICES Y SEGMENTACION	12
FUNCIONES	11
★★	10
- DE SIGNO	9
★	8
/	8
+	6
—	6
NOT	4
AND	3
OR	2

Veamos cómo actúan en realidad estos operadores para comprenderlos mejor. Cuando una relación es verdadera, toma el valor 1 y cuando es falsa, el valor 0. Por ejemplo, pruebe PRINT 7 ≠ 5 y PRINT 7 = 5. En el primer caso se imprimirá un 1, puesto que la relación es verdadera y, en el segundo caso, se imprimirá un 0, puesto que la relación es falsa.

Así pues, en la instrucción IF «condición» THEN «sentencia» se ejecuta la sentencia siempre que el valor de la condición sea distinto de 0.

Ejemplo: En IF 3 THEN PRINT "HOLA" siempre se ejecutará la impresión de "HOLA" pues $3 \neq 0$.

Visto esto, la actuación de los operadores lógicos, cuando operen con relaciones, se pueden resumir en unas tablas.

R1	R2	AND
1	0	0
1	1	1
0	1	0
0	0	0

R1	R2	OR
1	0	1
1	1	1
0	1	1
0	0	0

R1	NOT
1	0
0	1

De un modo más general, podemos pensar que AND, OR y NOT son operaciones con valor numérico, pues actúan de la siguiente forma:

Sean E1 y E2 expresiones numéricas cualesquiera, se tiene que:

$$E1\ AND\ E2 = \begin{cases} E1 \text{ si } E2 \neq 0 \\ 0 \text{ si } E2 = 0 \end{cases}$$

$$E1\ OR\ E2 = \begin{cases} 1 \text{ si } E2 \neq 0 \\ E1 \text{ si } E2 = 0 \end{cases}$$

$$NOT\ E1 = \begin{cases} 0 \text{ si } E1 \neq 0 \\ 1 \text{ si } E1 = 0 \end{cases}$$

Ejemplo de aplicación: El BASIC del ZX81 no tiene la instrucción ON expresión GOTO número de línea, número de línea... número de línea que actúa del siguiente modo: se calcula la expresión y si ésta da por ejemplo 4, salta al cuarto número de línea especificada en la lista. Para resolver este mismo problema con el ZX81, podemos utilizar lo que hemos visto del siguiente modo: supongamos que quere-

mos construir una instrucción que sea equivalente a: ON A GOTO 100, 525, 1030, 15. Esto se resuelve del siguiente modo:

GOTO (100 AND A = 1) + (525 AND A = 2) + (1030 AND A = 3) + (15 AND A = 4).

La operación con AND es la única operación lógica que se puede construir con cadenas y actúa del siguiente modo:

«CADENA» AND (Número) = { CADENA si el número es distinto de 0
" " (la cadena vacía) si el número es 0

CAPITULO 9

PROGRAMACION CON BUCLES

Supongamos que tenemos un programa con varios datos de entrada a los que hay que hacerles un tratamiento que es el mismo para todos los datos. Para ello no hace falta hacer un trozo de programa distinto para cada dato. Puede ser el mismo para todos, teniendo en cuenta que hay que cambiar de dato cada vez que termina el proceso. A este tipo de procesos repetitivos se les llama *bucles.* Veamos un ejemplo:

```
   10 LET TOTAL=0
   20 PRINT "ESCRIBA UN NUMERO"
   30 INPUT NUM
   40 CLS
   50 LET TOTAL=TOTAL+NUM
   70 PRINT "TOTAL ACUMULADO:";TO
TAL
   60 GOTO 20
```

Instrucción FOR NEXT

Ya sabemos en qué consiste un bucle y cómo hacer uso de él en un programa. Lo que ahora nos interesa es un tipo especial de bucle que nos permite hacer el BASIC del ZX81: el bucle FOR NEXT.

Un bucle FOR NEXT sólo se diferencia del que ya hemos visto anteriormente, en que tiene una variable de control que cuenta las veces que éste se ejecuta y que permite limitarlo a un número de ejecuciones predeterminado. La variable de control o contador también permite que a cada vuelta se realice un proceso ligeramente distinto si lo hacemos intervenir en él (esto se verá claramente en un ejemplo posterior).

La instrucción FOR NEXT tiene el siguiente formato:

FOR (variable de control) = (valor inicial) TO (límite) STEP (incremento). Esto se podría traducir del siguiente modo:

«Desde que la variable de control vale el valor inicial, hasta que vale el límite, aumentándole cada vez el incremento». A efectos de explicación, la abreviaremos del siguiente modo FOR V = I TO L STEP P.

V es el nombre de variable que sólo puede tener una letra.

I es el primer valor de V; puede ser cualquier expresión.

L es el último valor de V; también puede ser cualquier expresión.

P es el paso o incremento; puede ser una expresión.

El final del bucle se señala con la instrucción NEXT V y la combinación de estas dos instrucciones actúa de la siguiente manera:

1.° Al pasar la primera vez por la instrucción FOR, se crea una variable especial y distinta de las normales, ya que, aparte del nombre y del valor actual, contiene también el valor inicial, el límite y el incremento.

2.° Se ejecutan todas las instrucciones siguientes hasta llegar a la instrucción NEXT.

3.° La instrucción NEXT hace que a la variable de control le sea sumado el valor del incremento y luego salta a la línea siguiente de la instrucción FOR.

4.° Este proceso se realiza hasta que la variable de control, al pasar por NEXT alcanza un valor superior al límite (o inferior, si el incremento es negativo), en cuyo caso se ejecuta la instrucción siguiente a NEXT.

El número de veces que se ejecuta el bucle se puede calcular mediante la siguiente expresión: número de veces = INT ([L — I]/P) + 1).

```
  10 PRINT "   ESTE PROGRAMA SAB
E CONTAR"
  20 PRINT "DESDE QUE NUMERO?"
  30 INPUT INICIO
  40 PRINT "HASTA QUE NUMERO?"
  50 INPUT LIMITE
  60 PRINT "INCREMENTO?"
  70 INPUT INCREMENTO
  80 CLS
  90 FOR I=INICIO TO LIMITE STEP
 INCREMENTO
 100 PRINT I
 110 NEXT I
 120 PRINT "            ----------
---"
 130 PRINT " HE CONTADO ";1+INT
 ((LIMITE-INICIO)/INCREMENTO);"
NUMEROS"
```

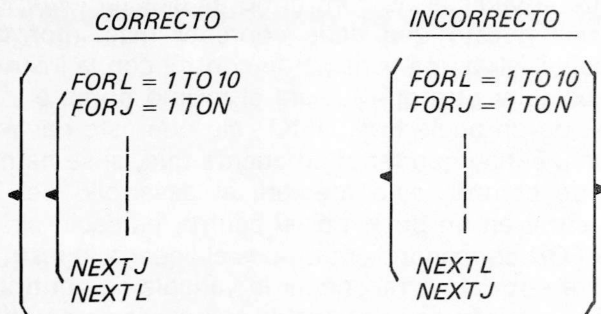
Observación: no es necesario que los valores iniciales, el límite y el STEP sean números enteros. Pueden ser perfectamente números decimales o incluso negativos.

En este ejemplo se compara un bucle FOR NEXT con un bucle construido con la instrucción GOTO:

```
  10 PRINT "ESCRIBA UN NUMERO"
  20 INPUT N
  30 LET SUM=0
  40 LET I=1
  50 LET SUM=SUM+1
  60 LET I=I+1
  65 IF I>N THEN GOTO 75
  70 GOTO 50
  75 CLS
  80 PRINT "LA SUMA DE LOS NUMER
OS DESDE EL 1 HASTA EL ";N;" ES
";SUM
```

```
  10 REM CALCULO DEL FACTORIAL D
E UN NUMERO
  15 CLS
  20 PRINT "NUMERO ?"
  30 INPUT NUM
  40 IF (NUM-INT NUM<>0) OR NUM
>33 THEN GOTO 200
  50 LET FACT=1
  60 IF NUM=0 THEN GOTO 100
  70 FOR I=NUM TO 1 STEP -1
  80 LET FACT=FACT*I
  90 NEXT I
 100 PRINT "EL FACTORIAL DE ";NU
M;" ES ";FACT
 110 GOTO 210
 200 PRINT "NUMERO INCORRECTO"
 210 PRINT ,,,,"DESEA PROBAR DE
NUEVO ?"
 220 INPUT A$
 230 IF A$>="S" THEN GOTO 15
 240 CLS
 250 PRINT ,,,,,,,,,,,,,,,,,,,,,,
"                 FIN"
```

Téngase en cuenta que dos bucles no pueden estar solapados, es decir, tienen que estar separados completamente o bien uno dentro del otro.



La razón de esto se hace evidente en el momento en que se intente seguir el flujo del bucle marcado como incorrecto pues nos encontramos con una instrucción FOR que inicializa un nuevo bucle pero seguida de una instrucción NEXT que hace volver al principio del bucle anterior con lo que el segundo no se ejecuta correctamente. Esto aún sería peor en el caso de que la variable de control se utilizara dentro del bucle interviniendo en las operaciones.

Veamos dos ejemplos correctos de utilización de un bucle FOR NEXT. Uno, el primero, en el que interviene la variable de control. Esta puede intervenir como dato pero hay que tener en cuenta que si se modifica, no se ejecutará el bucle correctamente. En el segundo programa, la variable de control no interviene y sólo se usa para contar el número de veces que se ejecuta el proceso.

```
  10 PRINT "ESCRIBA UN NUMERO"
  20 INPUT N
  30 LET SUM=0
  40 FOR I=N TO 1 STEP -1
  50 LET SUM=SUM+I
  60 NEXT I
  65 CLS
  70 PRINT "LA SUMA DE LOS NUMER
OS DESDE EL 1 HASTA EL ";N;" ES
";SUM
```

```
  10 PRINT "ESCRIBE UNA PALABRA"
  15 INPUT V$
  20 PRINT "CUANTAS VECES LA REP
ITO ?"
  30 INPUT V
  40 FOR I=1 TO V
  50 PRINT V$;",□";
  60 NEXT I
```

Recuérdese que la variable de control no es exactamente igual a una variable normal puesto que debe contener más información. Por ello cuando se inicializa una variable de control con la instrucción FOR se elimina cualquier otra que tuviera el mismo nombre.

Es posible salir de un bucle FOR NEXT, sin que éste haya terminado. Si se vuelve a él hay que tener en cuenta que, si se ha modificado la variable de control, esto afectará al desarrollo del bucle.

No se puede entrar en un bucle por el centro, es decir, sin pasar por la instrucción FOR correspondiente, pues al llegar a la instrucción NEXT, se producirá error al no reconocer la variable de control. Esto ocurrirá incluso si se ha definido una variable con el mismo nombre que en NEXT ya que no tendrá las características de una variable de control.

Sólo se puede entrar en medio de un bucle si es uno del cual se ha salido anteriormente ya que entonces la variable de control ha sido definida. Aún así hay que tener cuidado y no es aconsejable entrar en un bucle por el centro.

CAPITULO 10

CONJUNTOS, VECTORES Y MATRICES

Cuando se explicaron en su momento las distintas clases de variables, se omitió deliberadamente un tipo: las llamadas variables dimensionadas o conjuntos.

Para comprobar la necesidad que existe de ellas podemos imaginar el siguiente caso: supongamos que tenemos que hacer un programa que nos proporcione las sumas acumuladas de un conjunto de 10 números, que son datos de entrada. Una forma de hacerlo es dar un nombre de variable distinto para cada número y luego ir sumándolos, presentando los resultados del primero, del primero más el segundo, del primero más el segundo más el tercero, etc., hasta la suma total y tendremos así las diez sumas acumuladas. Este método es muy pesado pues los números pueden ser tanto 10 como 50 ó 200.

Nuestro problema quedaría resuelto si pudiéramos hacer el programa del siguiente modo:

```
10 LET SUM=0
20 FOR I=1 TO 10
30 INPUT N I
40 NEXT I
50 LET SUM=0
60 FOR I=1 TO 10
70 LET SUM=SUM+N I
80 PRINT SUM
90 NEXT I
```

De manera que al ir variando I las variables que se introducen en la sentencia INPUT fueran distintas, es decir N1, N2, N3, ... hasta N10. Pero esto no ocurre así, sino que sólo existe una variable que es N1 a la que se entra 10 veces y que luego se suma 10 VECES PERO, UNICAMENTE, CON EL ULTIMO VALOR QUE SE HA INTRODUCIDO. Tenga en cuenta pues, que este programa no funciona correctamente.

Este problema queda solucionado gracias a la existencia de variables dimensionadas, que junto con los bucles FOR NEXT forman la combinación más potente de que dispone este BASIC.

Variables dimensionadas. Vectores

Podemos imaginar una variable dimensionada como un cajón con distintas separaciones, de manera que cada una de ellas contiene una variable normal y ésta es simplemente accesible citando el nombre del cajón y el número del departamento que la contiene.

El cajón es lo que se llama conjunto que contiene unos elementos que se identifican por números llamados subíndices.

Para tener un conjunto en el ZX81 previamente hay que reservar un espacio en la memoria, indicando el número de elementos máximo que va a contener el conjunto. Esto se hace con la instrucción DIM, cuyo formato es: DIM nombre (número de elementos). El nombre, al igual que en el caso de las variables FOR NEXT, debe ser de una sola letra y para obtener el elemento N-esimo del conjunto se debe escribir nombre (N).

Ejemplo: la instrucción DIM A (5) reserva en memoria un bloque de cinco celdas todas las cuales se llaman A y que se distinguen por el número que sigue a la letra.

A (1)	A (2)	A (3)	A (4)	A (5)

Cada letra independientemente es como una variable normal.

Al ejecutar la instrucción DIM, aparte de reservarse el espacio para el conjunto, quedan inicializados todos sus elementos con el valor 0. También se suprime cualquier otro conjunto que tenga el mismo nombre pero no una variable normal, puesto que no hay posibilidad de confundirla, ya que la variable dimensionada lleva siempre detrás el paréntesis con el subíndice.

Este subíndice puede ser el mismo, otra variable o incluso otra

expresión, mientras no exceda al número máximo expresado en la sentencia DIM, pues entonces provocará un error.

En este ejemplo vemos cómo se puede resolver el problema planteado a principio del Capítulo, empleando una variable dimensionada.

```
 5 DIM N(10)
10 LET SUM=0
20 FOR I=1 TO 10
30 INPUT N(I)
40 NEXT I
50 LET SUM=0
60 FOR I=1 TO 10
70 LET SUM=SUM+N(I)
80 PRINT SUM
90 NEXT I
```

Matrices

A este tipo de conjunto que hemos utilizado hasta ahora, se le llama unidimensional o vector pero también es posible emplear conjuntos de más de una dimensión, es decir, sus elementos vendrán definidos por dos subíndices en lugar de uno. Intuitivamente podemos pensar que

B (1, 1)	B (1, 2)	B (1, 3)	B (1, 4)
B (2, 1)	B (2, 2)	B (2, 3)	B (2, 4)
B (3, 1)	B (3, 2)	B (3, 3)	B (3, 4)
B (4, 1)	B (4, 2)	B (4, 3)	B (4,4)

Figura 15

lo que se consigue con una instrucción de tipo DIM B (4,4), es una tabla de 4 filas por 4 columnas en la que cada elemento se obtiene especificando como subíndice el número de fila y el número de columna (fig. 15).

En realidad no se almacena en forma de tabla, como veremos en el Capítulo 14, pero está pensado para que se utilice como tal.

Conjuntos

Todo esto se puede generalizar a más dimensiones. Así, por ejemplo, un conjunto de tres dimensiones podría ser representado mediante un cubo. Ejemplo: DIM A (3,2,4) se representa como en la fig. 16.

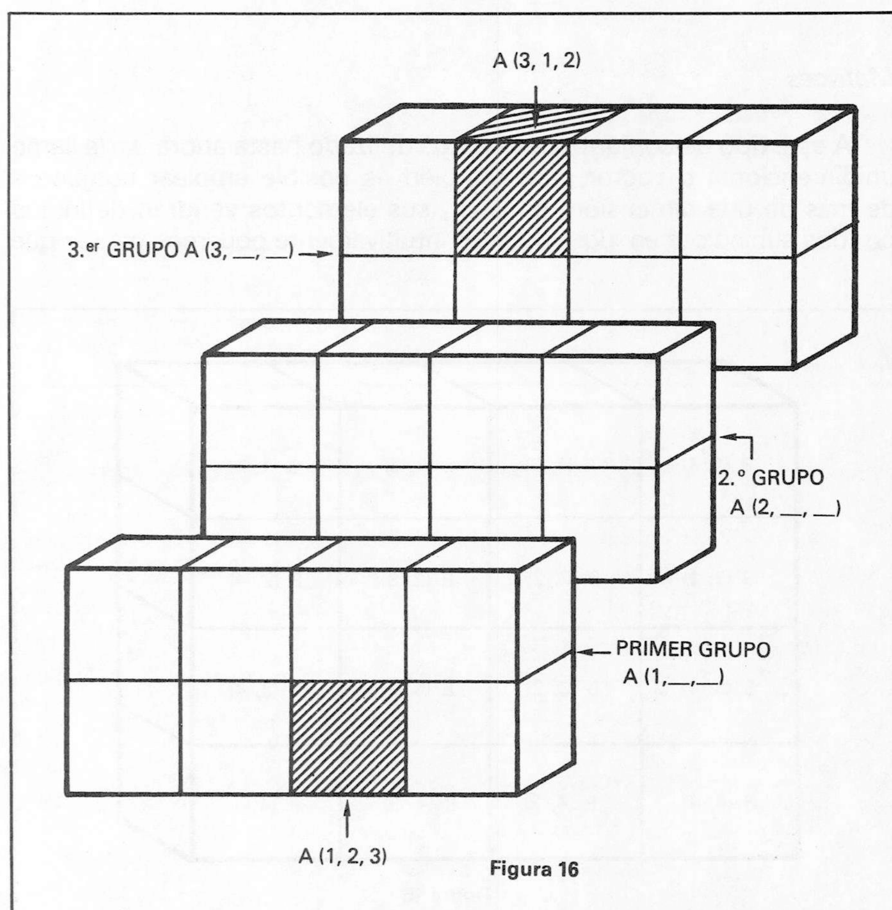


Figura 16

Un conjunto de más de tres dimensiones ya no es posible representarlo geométricamente pero podemos pensar, por ejemplo, en que tenemos varios libros, de modo que a cada uno le hemos designado un número para distinguirlo de los demás. La escritura de estos libros está en clave y cada letra viene representada por un número y en una misma página una letra se distingue de las otras, por la columna y fila en que se encuentra escrita. Esto puede ser representado mediante un conjunto de cuatro dimensiones en el que la primera indica el número del libro, la segunda el número de página, la tercera el número de fila y la cuarta el número de columna que corresponde al lugar que ocupa el carácter en la fila.

Así, si a este conjunto lo llamáramos A, su definición sería del siguiente modo: DIM A (número de libros, número de páginas por libro, número de filas por página, número de caracteres por fila).

Para localizar el tercer carácter de la fila 20, de la página 132 del cuarto libro, escribiremos: A (4, 132, 20, 3). Podemos así continuar empleando dimensiones al distinguir el estante en que se encuentra el libro, la sala en que se encuentra el estante, la biblioteca, la ciudad, etcétera.

Con el ZX81 no podríamos llegar a tanto pues al emplear conjuntos dimensionados el gasto de memoria crece muy rápidamente. Teniendo en cuenta que cada número real ocupa 5 bytes, el conjunto introducido por la instrucción DIM A (10, 150, 20, 3), ocuparía 450.000 bytes y el ZX81 sólo dispone, sin la ampliación de memoria, de 1024 bytes de los cuales más de 100 no son utilizables pues contienen a las variables del sistema, que serán estudiadas en el Capítulo 14.

Los conjuntos de dos dimensiones en matemáticas se llaman matrices y son útiles en muchos campos distintos, por ejemplo, en resolución de sistemas lineales.

Este programa es un ejemplo en el que se resuelve un sistema de ecuaciones compatible determinado. Para no hacerlo muy largo, se ha aplicado el método del pivotaje total en el que el error se propaga bastante. Este programa no cabe en 1 K.

```
  10 PRINT "RESOLUCION DE UN SIS
TEMA DE ECUACIONES LINEALES"
  15 PRINT
  16 PRINT "NUMERO DE INCOGNITAS
";
  20 INPUT N
  30 PRINT N
  33 PRINT
  35 DIM A(N,N)
```

```
36 DIM O(N)
37 DIM B(N)
38 DIM X(N)
40 FOR I=1 TO N
43 LET O(I)=I
50 FOR J=1 TO N
60 PRINT "A";I;J;"=";
70 INPUT A(I,J)
80 PRINT A(I,J),
81 NEXT J
82 NEXT I
83 PRINT
84 PRINT "TERMINOS INDEPENDIEN
TES"
86 FOR I=1 TO N
87 PRINT "B";I;"=";
89 INPUT B(I)
92 PRINT B(I)
95 NEXT I
100 PRINT ,,
101 LET S=0
102 FOR K=1 TO N-1
105 LET PIV=0
110 FOR L=K TO N
130 FOR M=K TO N
150 IF ABS A(L,M)>PIV THEN LET
F=L
160 IF ABS A(L,M)>PIV THEN LET
C=M
170 IF ABS A(L,M)>PIV THEN LET
PIV=A(L,M)
180 NEXT M
182 NEXT L
190 IF F<>K THEN GOSUB 2000
200 IF C<>K THEN GOSUB 1000
205 LET M=0
210 FOR I=K+1 TO N
212 PRINT
215 IF ABS A(K,K)<=1E-20 THEN G
OTO 2100
220 LET M=A(I,K)/A(K,K)
230 FOR J=K TO N
240 LET A(I,J)=A(I,J)-M*A(K,J)
250 NEXT J
253 LET B(I)=B(I)-M*B(I-1)
260 NEXT I
270 NEXT K
280 FOR K=N TO 1 STEP -1
290 LET SUM=0
300 FOR I=K+1 TO N
310 LET SUM=SUM+A(K,I)*X(O(I))
320 NEXT I
325 IF ABS A(K,K)<=1E-20 THEN G
OTO 2100
330 LET X(O(K))=(B(K)-SUM)/A(K,
K)
340 NEXT K
342 CLS
```

```
 345 PRINT
 346 PRINT "████████████████████████
████████████████"
 350 PRINT "SOLUCIONES DEL SISTE
MA:"
 355 PRINT "████████████████████████
████████████████"
 360 FOR I=1 TO N
 370 PRINT "X";I;"=";X(I)
 380 NEXT I
 390 STOP
1000 DIM V(N)
1010 FOR R=1 TO N
1020 LET V(R)=A(R,C)
1030 LET A(R,C)=A(R,K)
1040 LET A(R,K)=V(R)
1042 LET A=O(C)
1043 LET O(C)=O(K)
1045 LET O(K)=A
1050 NEXT R
1060 RETURN
2000 DIM V(N)
2010 FOR R=1 TO N
2020 LET V(R)=A(F,R)
2030 LET A(F,R)=A(K,R)
2040 LET A(K,R)=V(R)
2043 LET B=B(F)
2045 LET B(F)=B(K)
2047 LET B(K)=B(F)
2050 NEXT R
2060 RETURN
2070 STOP
2100 PRINT "PROBLEMA MAL CONDICI
ONADO: EL DETERMINANTE ESTA MUY
CERCANO O ES IGUAL A 0."
```

Para los que conozcan el método se proporciona la siguiente información sobre el programa.

A (N,N)	*Matriz del sistema*		
B (N)	*Vector de términos independientes*		
X (N)	*Vector de soluciones*		
V (R) y O (N)	*Vectores auxiliares para redondear la matriz y soluciones al efectuar el pivotaje*		
LINEAS 10—100	*Entrada de datos*		
LINEAS 101—270	*Triangulación por el método de pivotaje total:*		
	—líneas 102—182		*Cálculo del pivote*
	—líneas 190—200		*Envío a subrutinas de reordenación si es necesario.*
	—líneas 210—260		*Triangulación*
		215—	*Prueba para problemas mal condicionados*
LINEAS 280—340	*Cálculo de soluciones*		
LINEAS 342—390	*Impresión de soluciones*		
Subrutinas 1000 y 2000	*Reordenación de vectores*		

Este programa multiplica dos matrices cuadradas:

```
  10 PRINT "DIMENSION DE LAS MAT
RICES:";
  15 INPUT N
  17 PRINT N
  20 DIM A(N,N)
  30 DIM B(N,N)
  40 FOR I=1 TO N
  50 FOR J=1 TO N
  60 PRINT "A";I;J;"=";
  70 INPUT A(I,J)
  80 PRINT A(I,J),
  90 PRINT "B";I;J;"=";
  95 INPUT B(I,J)
 100 PRINT B(I,J),
 110 NEXT J
 120 NEXT I
 125 CLS
 130 FOR I=1 TO N
 140 FOR J=1 TO N
 150 LET S=0
 160 FOR K=1 TO N
 170 LET S=S+A(I,K)*B(K,J)
 180 NEXT K
 190 PRINT "R";I;J;"= ";S,
 200 NEXT J
 210 NEXT I
```

El ZX81 también tiene la posibilidad de definir conjuntos de cadenas. Estas se definen exactamente igual pero con un nombre de variable de cadena, por ejemplo: A$.

Una cadena que pertenezca a un conjunto se diferencia de una cadena normal, en que son de una longitud fija que viene determinada por la última especificación de dimensión. Así por ejemplo, la instrucción DIM A$ (4, 8), la podemos pensar, bien como un conjunto de 4 * 8 = 32 caracteres o bien como cuatro cadenas de longitud fija 8.

Si se asigna a una cadena de longitud fija otra de mayor longitud, ésta se corta por la derecha para que adquiera la longitud requerida, mientras que si la que asignamos es de menor longitud, se rellena con espacios a la derecha.

Los conjuntos de cadena poseen una particularidad que no tienen los de números y es que se puede suprimir el último subíndice tanto en el momento en que se está asignando (instrucción LET o INPUT), o cuando se quiere obtener (instrucción PRINT). Con esto lo que se consigue es que, en lugar de tener que tratar las cadenas de carácter en carácter, se pueden tratar como cadenas de longitud fija.

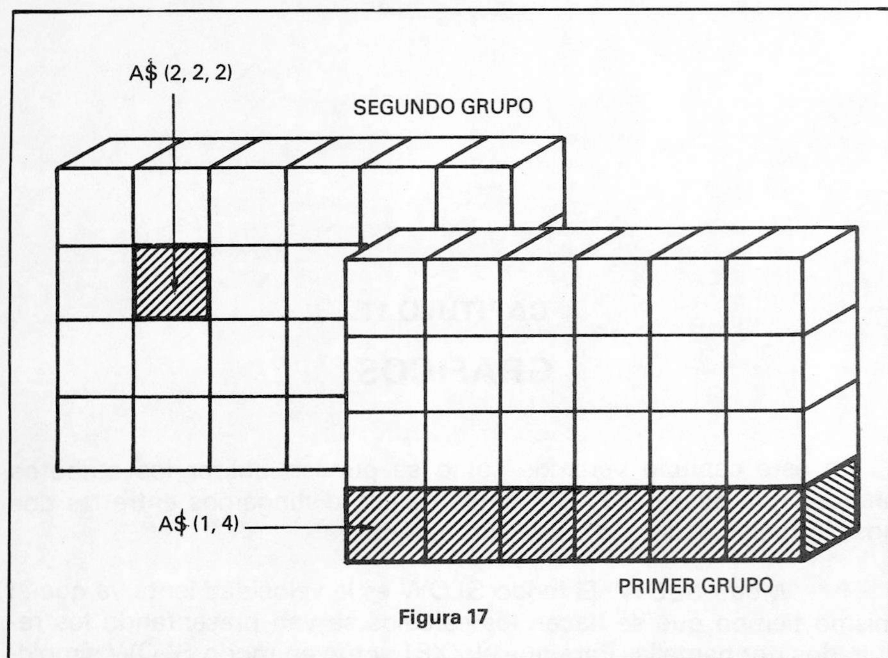


Figura 17

Ejemplo: sea la cadena definida por DIM A$ (2, 4, 10). Esto se puede representar gráficamente como en la figura 17.

Si se hace PRINT A$ (2, 2, 2), se obtiene el carácter segundo de la segunda cadena del segundo grupo, mientras que PRINT A$ (1, 4), proporciona toda la cuarta cadena del primer grupo.

El último subíndice también admite el segmentado. Así, por ejemplo, A$ (1, 2) (3 to 5) es lo mismo que A$ (1, 2, 3, to 5) y proporciona del tercer al quinto carácter ambos inclusive, de la segunda cadena del primer grupo.

CAPITULO 11

GRAFICOS

En este capítulo veremos cómo se pueden utilizar los atributos gráficos que posee el ZX81. Antes de ello distingamos entre las dos posibles velocidades de operación que posee:

1.° *Modo SLOW.* El modo SLOW es la velocidad lenta ya que al mismo tiempo que se hacen los cálculos se van presentando los resultados por pantalla. Para que el ZX81 actúe en modo SLOW simplemente hay que decírselo (SHIFT D).

2.° *Modo FAST.* El modo FAST es cuatro veces más rápido, debido a que no muestra el archivo de imagen si no es en uno de estos cuatro casos:

a) En una instrucción INPUT.
b) En una instrucción PAUSE.
c) Cuando la pantalla está llena.
d) Al final de programa o al detenerse debido a una instrucción STOP.

Tanto la instrucción SLOW como FAST, se pueden utilizar como líneas de programa. La instrucción SLOW es aconsejable durante aquellos trozos de programa en los que hay que representar resultados por pantalla y también en los programas de gráficos en los que resulta prácticamente indispensable. Por su parte, la instrucción FAST es muy indicada para los programas que tienen que realizar gran cantidad de cálculos.

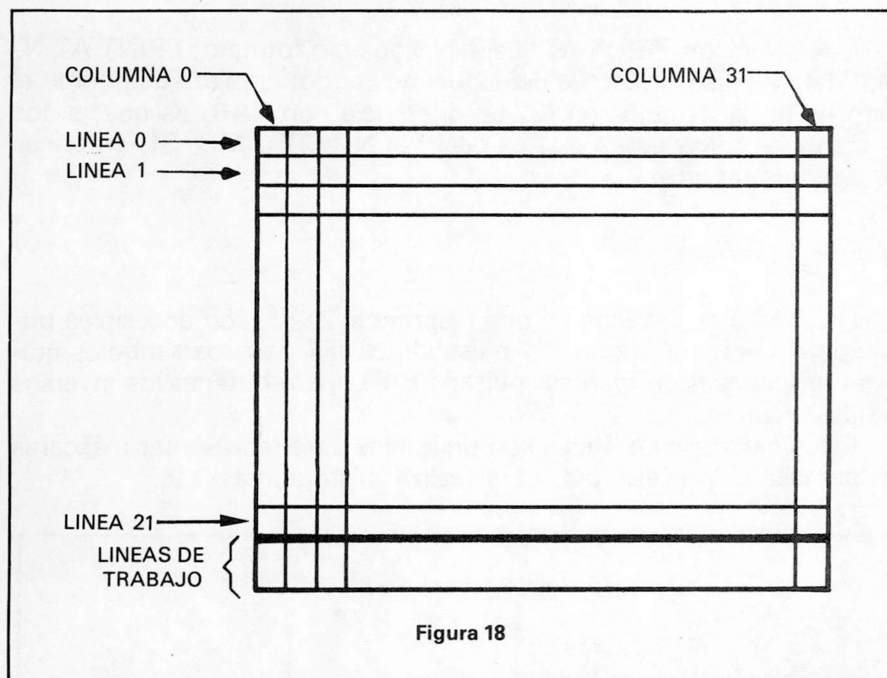


Figura 18

Sentencia TAB y PRINT AT

Al tratar de la instrucción PRINT, únicamente se ofreció la diferencia entre la separación por coma o por punto y coma, a la hora de imprimir los datos. Veamos cómo estas dos instrucciones nos permiten situar más claramente la posición de un dato a imprimir.

La sentencia PRINT TAB N; (dato a imprimir), sitúa la posición de PRINT en la columna N y escribe el dato.

Si el número N es mayor que 32 se divide por 32 y se sustituye por el resto. TAB siempre cuenta desde el principio, no desde la última posición y es útil para escribir datos columnados.

Instrucción PRINT AT

La pantalla del ZX81 está dividida en 24 cifras y 32 columnas, numeradas del 0 al 31 y del 0 al 23 respectivamente. Las dos últimas filas 22 y 23 no son utilizables para escribir en ellas y sirven para entrar datos y dar informes (fig. 18).

La instrucción PRINT AT, tiene el siguiente formato: PRINT AT N, M; "dato a imprimir. Actúa del siguiente modo: empieza a imprimir el dato en la fila N columna M. La diferencia con TAB, es que, si los números se salen de escala, es decir, si N > 21 ó M > 31, entonces se produce un error.

Símbolos gráficos

Los símbolos gráficos de que dispone el ZX81, son accesibles poniendo el cursor en modo G y pulsando SHIFT para los símbolos gráficos propiamente o bien sin pulsar SHIFT, para obtener los inversos de los caracteres.

Estos caracteres gráficos son utilizables para representar máscaras en pantalla o, por ejemplo, para realizar histogramas, etc.

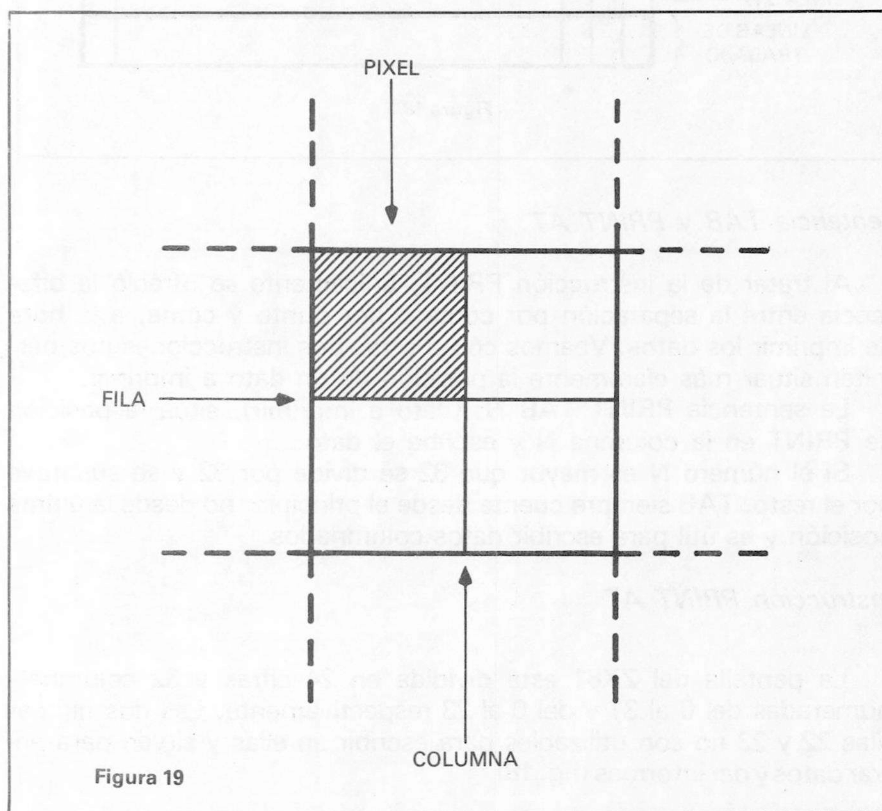


Figura 19

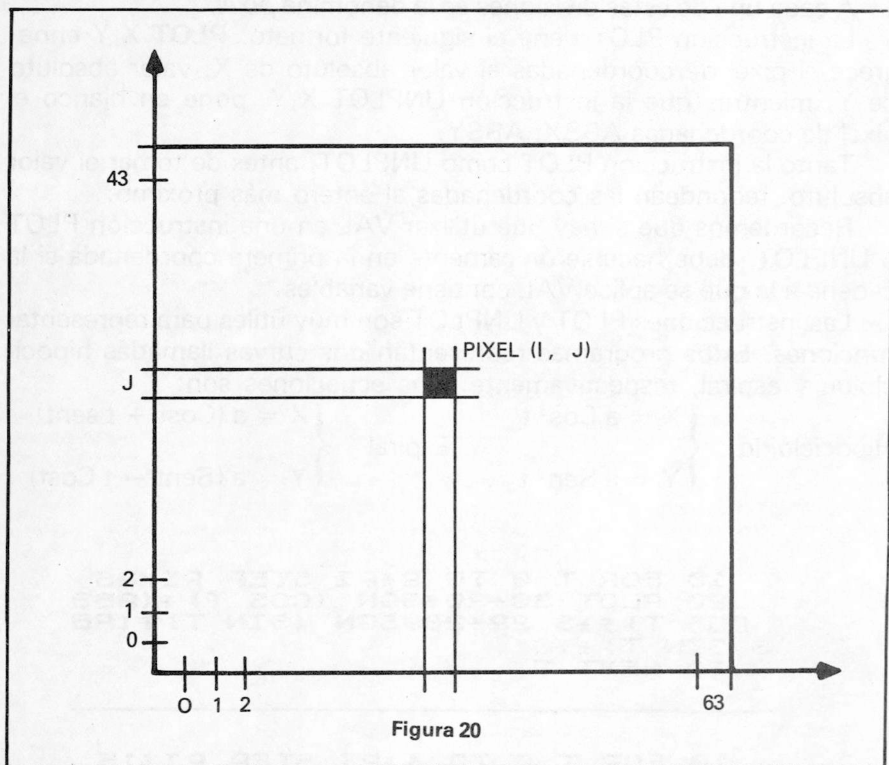


Figura 20

A lo largo del Capítulo se examinarán programas de gráficos.

Los caracteres gráficos se pueden asignar perfectamente a cadenas, lo cual las hace muy manejables.

Otras dos sentencias muy útiles a la hora de trabajar con gráficos son: PLOT y UNPLOT.

Para comprender lo que hacen las sentencias PLOT y UNPLOT debemos pensar en una nueva división de la pantalla. Cada uno de los cuadrados determinados por una fila y una columna de la división que habíamos hecho anteriormente se divide, a su vez, en cuatro zonas más (fig. 19) con lo que obtenemos 64 columnas (numeradas de la 0 a la 63) y 44 filas (numeradas de la 0 a la 43).

La diferencia con AT, es que aquí la situación es como en un eje de coordenadas en el que la primera se refiere a la columna mientras que la segunda se refiere a la fila y además éstas se cuentan de abajo arriba en lugar de hacerlo de arriba a abajo como en la sentencia PRINT (fig. 20).

A cada una de estas divisiones se la denomina *pixel.*

La instrucción PLOT tiene el siguiente formato: **PLOT** X,Y ennegrece el pixel de coordenadas el valor absoluto de X, valor absoluto de Y, mientras que la instrucción UNPLOT X,Y, pone en blanco el pixel de coordenadas ABSX, ABSY.

Tanto la instrucción PLOT como UNPLOT, antes de tomar el valor absoluto, redondean las coordenadas al entero más próximo.

Recordemos que si hay que utilizar VAL en una instrucción PLOT o UNPLOT, debe hacerse únicamente en la primera coordenada si la cadena a la que se aplica VAL contiene variables.

Las instrucciones PLOT y UNPLOT son muy útiles para representar funciones. Estos programas representan dos curvas llamadas hipocicloide y espiral, respectivamente. Sus ecuaciones son:

Hipocicloide $\begin{cases} X = a \, \mathrm{Cos}^3 \, t \\ Y = a \, \mathrm{Sen}^3 \, t \end{cases}$ Espiral $\begin{cases} X = a \, (\mathrm{Cos}t + t \, \mathrm{sen}t) \\ Y = a \, (\mathrm{Sen}t - t \, \mathrm{Cos}t) \end{cases}$

```
   10 FOR T=0 TO 2*PI STEP PI/25
   20 PLOT 30+20*SGN (COS T)*(ABS
 COS T)**3,22+20*SGN (SIN T)*(AB
S SIN T)**3
   30 NEXT T
```

```
   10 FOR T=0 TO 4*PI STEP PI/15
   20 PLOT 33+2*(COS T+T*SIN T),2
4+2*(SIN T-T*COS T)
   30 NEXT T
```

Nótese que para elevar CosT al cubo se ha tenido que separar de su signo puesto que el ZX81 no permite las potencias de números negativos.

En lo que sigue se utilizan los conceptos estudiados anteriormente para desarrollar las posibilidades de juegos gráficos con el ZX81.

Sólo se darán unas ideas básicas para que el lector, a partir de ellas, pueda desarrollar programas de mayor envergadura.

Juegos gráficos

El uso combinado de INKEY$ y PLOT, nos permite construir un programa que dibuje en la pantalla.

```
  10 LET X=32
  20 LET Y=34
  30 IF INKEY$="" THEN GOTO 30
  40 LET A$=INKEY$
  50 LET X=X+(1 AND A$="8")-(1 A
ND A$="5")
  60 LET Y=Y+(1 AND A$="7")-(1 A
ND A$="6")
  70 PLOT X,Y
  80 GOTO 30
```

Este programa dibuja en la pantalla, en la dirección de las flechas correspondientes a las teclas 5, 6, 7 y 8. El primer punto «ploteado» se puede variar a voluntad (líneas 10 y 20). La línea 30 se espera hasta que una tecla es pulsada, la línea 40 asigna el valor de la tecla a la variable A$ y las líneas 50 y 60 calculan la modificación que se tiene que hacer a las coordenadas X e Y según la tecla que se ha pulsado. Esto se realiza utilizando los operadores lógicos. Por último, la línea 70 dibuja el punto cuyas coordenadas se han calculado.

Un buen ejercicio sería realizar un programa que borre, además de dibujar. Claro está que se deberá utilizar la instrucción UNPLOT.

La instrucción PAUSE se puede utilizar como temporizador, ya que actúa del siguiente modo: el valor que se le indica se almacena en dos posiciones de memoria (la variable FRAMES, que se verá en el Capítulo dedicado a las variables del sistema). Este valor se va disminuyendo a razón de 50 unidades cada segundo por lo que nos puede servir para construir un reloj.

```
   2 PRINT AT 8,5;"HORA:";
   4 INPUT H
   6 PRINT H
   8 PRINT AT 11,5;"MINUTOS:";
  10 INPUT M
  12 PRINT M
  14 PRINT AT 20,NOT PI;"PULSE U
NA TECLA PARA EMPEZAR"
  16 PAUSE 35000
  17 CLS
  18 PRINT AT 9,11;"▛▀▀▀▀▀▀▀▀▜"
  20 PRINT AT 10,11;"▌  :  :  ▐"
  22 PRINT AT 11,11;"▙▄▄▄▄▄▄▄▄▟"
  40 FOR K=H TO 23
  45 PRINT AT 10,12;K
  50 FOR J=M TO 59
  55 PRINT AT 10,15;J
  60 FOR I=1 TO 59
  65 PAUSE 27
  70 PRINT AT 10,18;I
  80 NEXT I
```

```
 85 PRINT AT 10,18;"0 "
 90 PRINT AT 10,15;J
 95 LET M=NOT PI
100 NEXT J
110 PRINT AT 10,15;"0 "
120 LET H=M
130 NEXT K
```

La instrucción PAUSE de la línea 65 es la que pone el reloj a punto con la máxima exactitud posible pero tiene el problema de que la pantalla «rebota» cada vez que se pasa por ella. Esto se puede solventar sustituyéndola por un bucle FOR...NEXT que pierda el mismo tiempo. Pruebe a cambiarla por:

```
62 FOR D=1 TO 21
64 NEXT D
```

Ejercicio: Intentar construir el mismo reloj pero con esfera, de manera que aparezcan los segundos girando en forma de un punto «ploteado».

La función RND junto con RAND, proporciona una manera de montar juegos de adivinanzas, en los que se puede marcar la serie del número pensado por RND.

Este programa pide un número, lo compara con el que se ha pensado y nos dice si es mayor o menor.

```
 10 RAND 0
 20 LET N=INT (1000*RND)
 30 PRINT AT 5,5;"ESCRIBAUN NUM
ERO"
 50 INPUT A
 55 CLS
 60 IF N>A THEN LET A$="MAYOR"
 70 IF N<A THEN LET A$="MENOR"
 80 IF N=A THEN GOTO 120
 90 PRINT AT 10,0;"EL QUE YO HE
PENSADO ES ";A$
110 GOTO 30
120 CLS
130 PRINT AT 10,7;"CORRECTO"
```

Ejemplo: En este programa el ordenador piensa un número de cuatro cifras y pide números, indicando luego el número de cifras que están en su sitio correcto y las que se han adivinado pero no están en su lugar. Este juego es conocido con el nombre de «Mastermind».

```
  1 PRINT "NUMERO";TAB 10;"DIAN
AS";TAB 20;"APROX."
  2 PRINT "[redacted]   [redacted]
[redacted]"
 10 DIM A$(4)
 20 LET A$="0"
 30 RAND VAL A$
 40 LET A$=STR$ (10000*RND)
 45 IF VAL A$<1234 THEN GOTO 30
 50 FOR I=1 TO 4
 60 FOR J=I+1 TO 4
 70 IF A$(I)=A$(J) THEN GOTO 30
 80 NEXT J
 90 NEXT I
 95 LET G=1
100 INPUT B$
101 DIM A(4)
102 DIM D(4)
103 LET A=0
104 LET D=0
110 IF LEN B$<>4 THEN GOTO 100
130 FOR J=1 TO 4
140 IF A$(J)=B$(J) THEN LET D(J
)=J
150 IF A$(J)=B$(J) THEN LET D=D
+1
160 NEXT J
180 FOR I=1 TO 4
185 FOR J=1 TO 4
194 IF B$(I)=A$(J) AND I<>J THE
N GOSUB 1000
200 NEXT J
210 NEXT I
250 PRINT B$;TAB 12;D;TAB 22;A
260 IF B$=A$ THEN GOTO 2000
270 LET G=G+1
280 IF G>15 THEN PRINT "SE LE H
AN TERMINADO LAS JUGADAS"
290 IF G>15 THEN STOP
300 GOTO 100
1000 FOR K=1 TO 4
1005 IF D(K)=0 THEN NEXT K
1007 IF K>4 THEN GOTO 1030
1010 IF B$(D(K))=B$(I) THEN RETU
RN
1015 NEXT K
1030 FOR R=1 TO 4
1035 IF A(R)=0 THEN NEXT R
1037 IF R>4 THEN GOTO 1060
1040 IF B$(A(R))=B$(I) THEN RETU
RN
1050 NEXT R
1060 LET A(I)=I
1070 LET A=A+1
1080 RETURN
2000 PRINT ,,,,TAB 12;"MUY BIEN"
```

A partir de este último programa es fácil realizar otros, como puede ser el «Juego del ahorcado», en el que se introduce una palabra que hay que adivinar, letra a letra, con un número limitado de tiradas. Su estructura será parecida pero computando cadenas en lugar de números.

CAPITULO 12

LA IMPRESORA

El ZX81 dispone opcionalmente de una impresora térmica, capaz de obtener más alta resolución que la pantalla mediante una rutina en código máquina que se incluye en el manual.

Las instrucciones que hacen uso de la impresora son las siguientes:

LPRINT que actúa exactamente igual que PRINT, sólo que lo que le sigue se imprime en la impresora.

Se puede utilizar TAB, pero si se utiliza AT, sólo tendrá efecto el número de columna.

LLIST actúa exactamente igual que LIST, listando el programa completo en la impresora.

COPY, vuelca el contenido de la pantalla en la impresora.

La impresora dispone de una memoria intermedia que corresponde a una línea de impresión situada en la dirección de memoria 16444 hasta 16476 (consúltese el Capítulo de las variables del sistema).

CAPITULO 13

EL ALMACENAMIENTO DE DATOS EN CASSETTE

A estas alturas se habrá ya comprobado que cuando se desconecta la corriente, el ZX81 pierde su información. La única manera de retenerla es grabándola en cinta. Para ello es preciso proveerse de un magnetófono.

En teoría, cualquiera sirve, pero en la práctica nò es así. Por discreción no se citarán marcas pero se puede consultar con otras personas o con algún club de usuarios.

Es aconsejable que el magnetófono sea monoaural y también resultará muy útil que disponga de micrófono incorporado para decir los nombres de los programas. Asimismo es interesante el contador de cinta para que los programas puedan ser localizados rápidamente al saber su número.

• Por supuesto, el magnetófono deberá tener entrada MIC, para grabar y EAR para altavoz externo y, si es posible, que estén ya adaptados a los jacks de 3,5 mm que se acompañan con el ZX81.

La grabación de un programa en cinta

Para grabar un programa en cinta se deben realizar las siguientes operaciones:

1.° Conectar la salida MIC del ZX81 con la entrada de grabación del magnetófono.

2.° Situar la cinta en la posición en la que se quiere grabar el programa.

3.° Si así se desea, grábese con su propia voz el nombre del programa.

4.° Pulse SAVE «nombre». Es «nombre» cualquiera que quiera

ponérsele al programa. No está permitido grabar un programa sin nombre.

5.º Ponga en marcha el magnetófono, en grabación.
6.º Pulse NEW LINE.

Se observará lo siguiente: durante unos cinco segundos se verá la pantalla de un color grisáceo.

Seguidamente aparecerán unas líneas negras horizontales durante un cierto tiempo que depende de la longitud del programa. Finalmente se reflejará en la pantalla el informe 0/0. Esta imagen característica que ofrece el televisor se debe a que éste recibe la misma señal que el enchufe MIC.

Para comprobar que todo ha ido bien se rebobinará la cinta y se comprobará si se ha grabado el sonido característico.

En primer lugar se oirá, si la ha grabado, su propia voz diciendo el nombre del programa, luego seguirá un zumbido que corresponde al tiempo transcurrido entre el momento en que se ha puesto en marcha el magnetófono y el que se ha pulsado NEW LINE. Después vendrán unos cinco segundos de silencio que corresponden al tiempo en que la pantalla del televisor estaba de color grisáceo, seguidos de un ruido estridente y agudo que es la grabación del programa. Finalmente se repetirá el zumbido del principio, que corresponde al tiempo transcurrido entre la terminación del programa y la detención del magnetófono.

Carga de un programa desde la cinta al ZX81

Para recuperar el programa grabado, se realizarán las siguientes operaciones:

1.º Se situará la cinta al principio de la grabación.
2.º Se escribirá LOAD «nombre del programa» (sin pulsar NEW LINE).
3.º Se pondrá en marcha el magnetófono, en reproducción.
4.º Se pulsará NEW LINE.

Si todo esto se desarrolla correctamente deberá aparecer en la pantalla un conjunto de rayas blancas y negras más gruesas que en el caso SAVE y al final aparecerá el informe 0/0.

Problemas que se pueden presentar a la hora de recuperar el programa:

1.º El volumen de reproducción tiene que ser lo suficientemente alto para que la señal sea captada correctamente pero no demasiado pues puede distorsionar, interrumpiéndose la grabación.

Lo más aconsejable, al principio, es hacer pruebas de volumen con programas cortos y cuando se encuentra el volumen adecuado, dejarlo fijo. También es conveniente, si es posible, colocar los agudos al máximo y los bajos al mínimo.

2.º Desde el principio puede interrumpirse la grabación. Esto puede ser debido a que el primer zumbido sea demasiado fuerte y el ZX81 lo interprete como si empezara el programa. La solución a esto consiste únicamente en situar la cinta en la parte silenciosa, entre el zumbido y el programa, antes de empezar la operación.

3.º El ZX81 puede no recibir correctamente el programa. Esto puede ser debido a dos causas: la primera, un volumen de reproducción defectuoso, que ya se ha comentado; la segunda, es que exista un desplazamiento del cabezal de grabación del magnetófono con respecto al de reproducción que se solventa graduándolo correctamente.

4.º También es posible que si tiene conectadas las clavijas de grabación y reproducción se produzca un bucle de realimentación. Esto se puede arreglar manteniendo conectada solamente la clavija que se utiliza en aquel momento.

5.º Se utilizarán cintas de calidad o bien se efectuarán las grabaciones por duplicado puesto que, con el tiempo, se puede desgastar la cinta impidiendo la recuperación de los programas. Cualquier interrupción en la grabación, por pequeña que sea, producirá la interrupción de la recuperación del programa, en el momento que ésta ocurra.

Un programa no se puede grabar sin un nombre. Si se intenta hacer SAVE " ", se producirá un error F.

Si al intentar recuperar un programa se da un nombre que no corresponde exactamente al que se dio cuando fue grabado, al ZX81, se intentará cargar el programa siguiente de la cinta comprobando previamente si se corresponden los nombres y así sucesivamente.

Si en el momento de recuperar un programa no se recuerda su nombre exacto, se puede dar como nombre la cadena vacía (" "). Entonces se carga el primer programa que se encuentra.

Autoejecución de un programa

SAVE se puede utilizar en programa y actúa del siguiente modo: cuando el programa pasa por la instrucción SAVE, ésta se graba en cinta en caso de que el magnetófono esté en marcha; si el magnetófono no está en marcha, se envía la señal a la salida MIC y a la TV. Después de esto, continúa la ejecución por la línea siguiente.

Si un programa que ha sido conservado utilizando la instrucción SAVE dentro del mismo programa se vuelve a grabar en el ZX81, se ejecuta automáticamente a partir de la línea siguiente a SAVE.

Cuando se haya hecho LOAD, se observará que el último carácter del nombre en la instrucción SAVE ha pasado a inverso pero esto no debe preocupar ya que es simplemente por razones de identificación.

CAPITULO 14

ESTRUCTURA DE LA MEMORIA

En el ZX81, como en la mayoría de los ordenadores, la memoria se puede entender como una larga fila de casillas, cada una de las cuales tiene un número para identificarla. Este número es conocido con el nombre de dirección o posición de memoria.

Cada una de las casillas contiene un byte, y, por lo tanto, cuando hablamos de una dirección de memoria, nos estamos refiriendo al valor del byte que contiene esa casilla. Un dibujo de esta estructura aparece en la Introducción de esta obra (Fig. 7).

El ZX81, tiene 8 K de ROM que corresponden a 8 × 1024 = 8192 bytes y son los primeros de la larga fila. Están numerados desde el 0 hasta el 8191. Después de ellos viene 1 K de RAM que es la memoria disponible para el usuario. Lo que ocurre es que la primera dirección de esas 1.024 que forman el K no es, como cabe esperar, la 8192 sino que empieza a contarse desde la 16.384. Con esto tenemos un salto de precisamente 8 K, que no se usan y que no existen físicamente aunque si se accede a ellos proporcionan una copia exacta de los 8 K de ROM.

La distribución de la RAM difiere ligeramente si se dispone de un módulo de ampliación de memoria (fig. 21).

Variables del sistema

Desde la dirección 16384 hasta 16508, se encuentran las variables del sistema. Estas variables reciben unos nombres que no son reconocidos por el ordenador y únicamente sirven a efectos nemotécnicos. En ellos están contenidos distintos tipos de información que es usada por las rutinas de la ROM para llevar a efecto su trabajo.

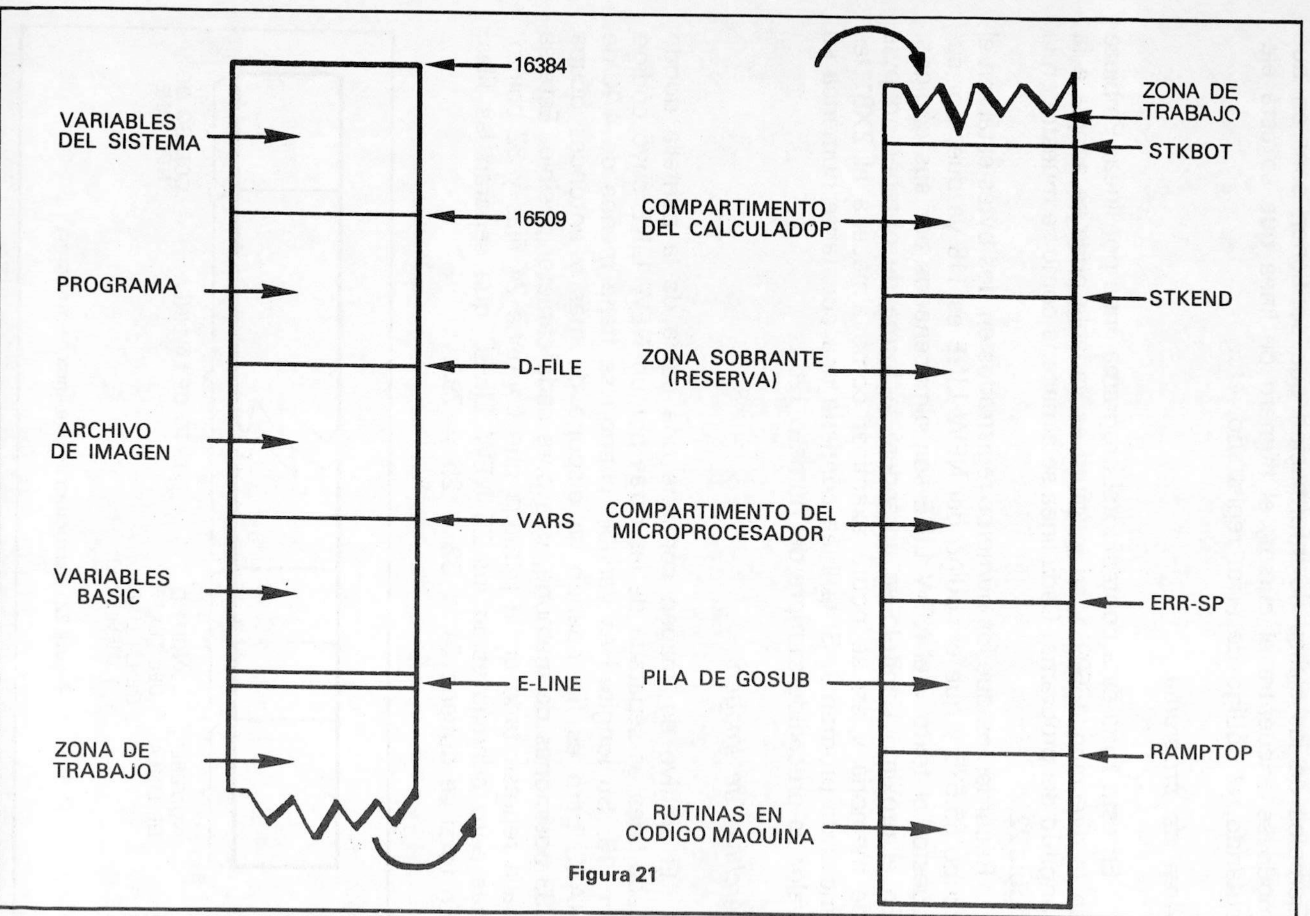

16384
VARIABLES DEL SISTEMA
16509
PROGRAMA
D-FILE
ARCHIVO DE IMAGEN
VARS
VARIABLES BASIC
E-LINE
ZONA DE TRABAJO
ZONA DE TRABAJO
STKBOT
COMPARTIMENTO DEL CALCULADOP
STKEND
ZONA SOBRANTE (RESERVA)
COMPARTIMENTO DEL MICROPROCESADOR
ERR-SP
PILA DE GOSUB
RAMPTOP
RUTINAS EN CODIGO MAQUINA

Figura 21

Ejemplos de las clases de información que contienen son: en qué modo se encuentra el cursor, el número de línea que se está ejecutando, el código de error registrado, etc.

Area de programa

En esta zona está contenido el programa línea por línea. Empieza en la dirección 16509 y su longitud es variable pues se acopla a la longitud del programa. Cada línea se almacena como se muestra en la figura 22.

Recordemos que los números registrados en dos bytes están en el rango 65.535 y que el código del NEW LINE es 118 ya que, por supuesto, el texto y el NEW LINE son almacenados por sus códigos. En el próximo Capítulo se aprenderá la forma de obtener un byte de memoria y así se podrá examinar cómo almacena el ZX81 las líneas de programa. Si la línea contiene una constante numérica su valor va precedido siempre del número 126.

Archivo de imagen

El archivo de imagen contiene una copia de la pantalla donde cada línea es separada de las otras por un NEW LINE cuyo código es 118. Su longitud es variable cuando se tiene menos de 4 K de RAM, pero es fija cuando se dispone de más y entonces ocupa 793 posiciones de memoria, una para cada carácter posible. Este se halla representado en la pantalla que contiene 24 filas y 32 columnas pero además están los 25 NEW LINES que separan las filas. En total se obtiene $24 \times 32 + 25 = 793$.

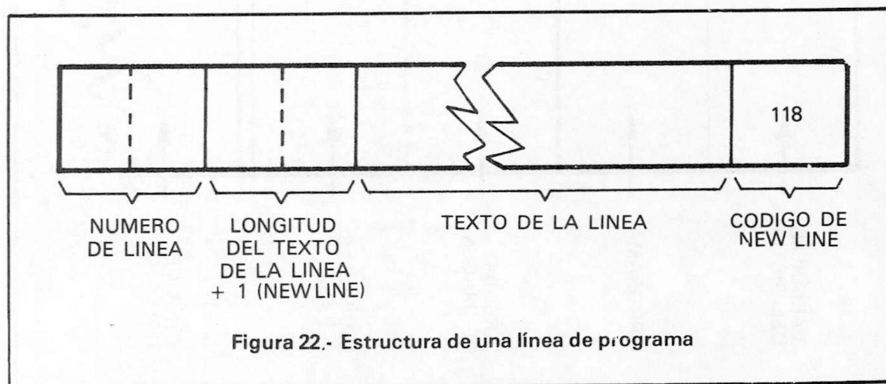


Figura 22.- Estructura de una línea de programa

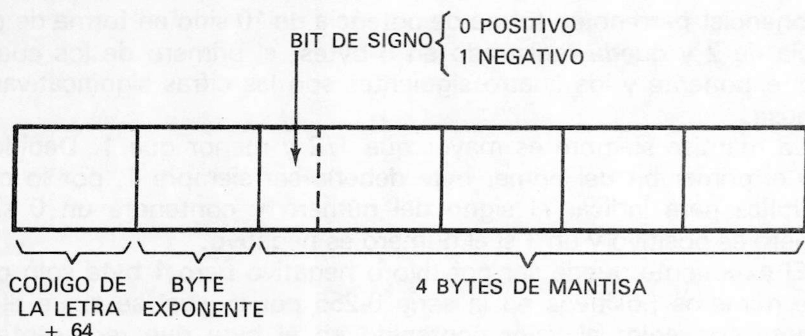


Figura 23.– Representación de una variable real cuyo nombre es de una sola letra

Cuando se dispone de menos de 4 K, el archivo se empieza conteniendo únicamente los 25 NEW LINES y cada una de las 24 líneas posibles es registrada entre los correspondientes NEW LINES de manera que las líneas que ocupan menos de 32 bytes no se gastan todas para almacenar como cuando se dispone de más memoria.

Se ha observado que la longitud de las zonas de memoria no es constante en muchos casos como ocurre, por ejemplo, cuando se disponen menos de 4 K. Pues bien, las direcciones en que empieza una zona y termina otra están registradas en algunas de las variables del sistema, cuyos nombres aparecen en la figura. Así, el archivo de imagen está situado entre la dirección registrada en la variable D— FILE y la registrada en VARS.

Area de variables

Las variables usadas en el programa o definidas mediante órdenes están almacenadas entre las direcciones VARS y E— LINE. La manera en que son almacenadas depende del tipo de variable que se reconoce en la forma en que queda registrado su nombre.

Veamos cómo se almacenan los distintos tipos de variables:

1.° *Variable numérica (real), cuyo nombre es de una sola letra* (figura 23).

El nombre de la variable se almacena como su código más 64. Así, por ejemplo, el nombre de la variable A queda registrado con el

número 102 = 38 + 64. El valor de la variable se almacena en forma exponencial pero no en forma de potencia de 10 sino en forma de potencia de 2 y queda registrado en 5 bytes, el primero de los cuales es el exponente y los cuatro siguientes son las cifras significativas o mantisa.

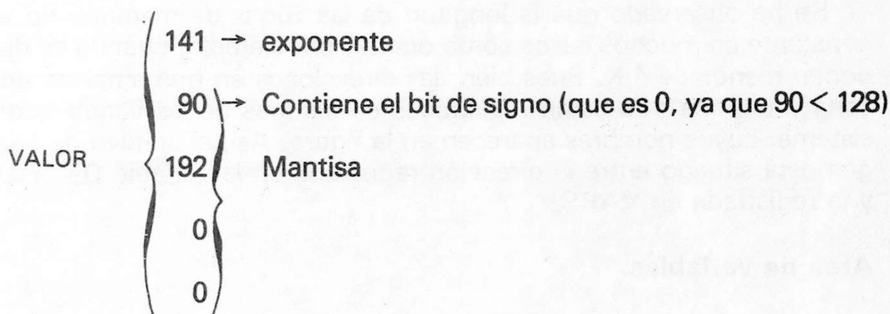
La mantisa siempre es mayor que 1/2 y menor que 1. Debido a esto el primer bit del primer byte debería ser siempre 1, por lo cual se utiliza para indicar el signo del número y contendrá un 0 si el número es positivo y un 1 si el número es negativo.

El exponente puede ser positivo o negativo pero 1 byte sólo permite números positivos en la serie 0-255 por lo cual se sigue el siguiente convenio: al valor contenido en el byte que representa al exponente se le resta 128 para obtener el verdadero exponente.

Así el valor de la variable se calcula del siguiente modo:
(S) M $\times$ 2^{E-128} donde S es el signo representado en el primer bit de la mantisa. M es la mantisa y E es el exponente.

Ejemplo:

LA INSTRUCCION LET A = 7.000 SE REFLEJA EN EL AREA DE VARIABLES COMO:
NOMBRE: 102 → 38 (Código de A) + 64 (Por ser variable numérica de una sola letra).



PARA RECUPERAR EL VALOR 7.000 A PARTIR DE ESTA EXPRESION SE PROCEDE:

Se suma a 90 porque el número es positivo

$$\left(\frac{0}{256^4} + \frac{0}{256^3} + \frac{192}{256^2} + \frac{90 + \boxed{128}}{256}\right) \times 2^{141 - 128} =$$

$= (0 + 0 + 0.00292968 + 0.8515625) \times 2^{13} =$

$= 0.8544921 \times 8192 = 7000$

LA INSTRUCCION LET A = −7000 PRODUCE:

102

141

218 → Al ser mayor que 128, quiere decir que el bit de signo contiene un 1 y, por tanto, que el número es negativo.

90

192

0

0

Para recuperar el valor se actúa de igual forma pero sin sumar 128 al primer byte de la mantisa.

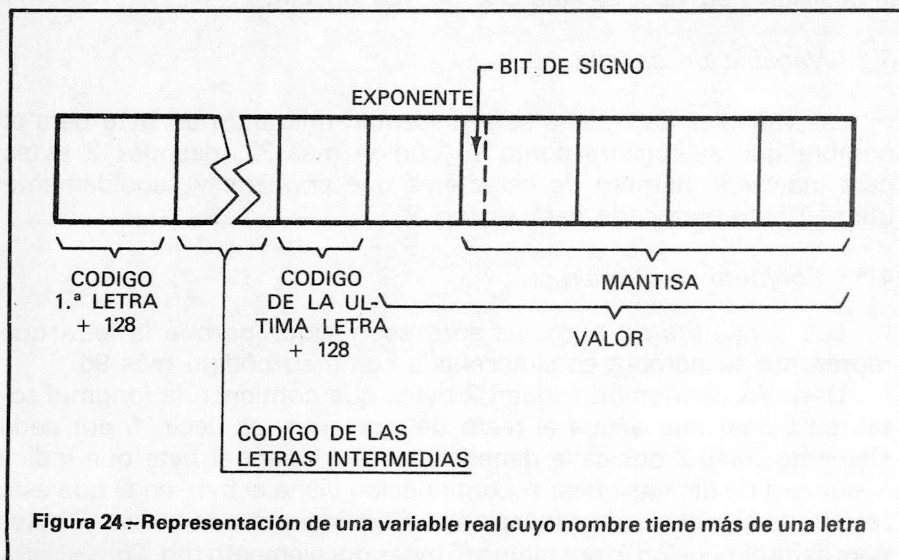


Figura 24.–Representación de una variable real cuyo nombre tiene más de una letra

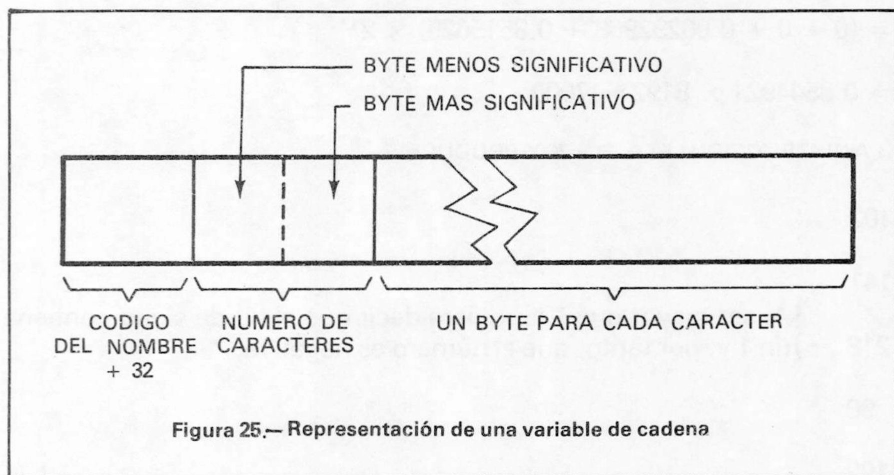


Figura 25.– Representación de una variable de cadena

2.º Variable cuyo nombre es de más de una letra

Este tipo de variables se representa de un modo muy similar a las anteriores con la única diferencia de que el nombre contiene más caracteres y a su código se le añade el valor 128. Esto quiere decir que el séptimo bit de cada uno de los bytes que representen al nombre, siempre contendrá un 1 ya que si lo evaluamos, al estar en la posición 7 su valor es igual a $2^7 = 128$ (fig. 24).

3.º Variable de cadena

Las variables de cadena se representan utilizando un byte para el nombre que se registra como su código más 32, después 2 bytes para indicar el número de caracteres que contiene y seguidamente utiliza 1 byte para cada carácter (fig. 25).

4.º *Conjunto de números*

Los conjuntos de números son reconocidos porque la letra que representa su nombre es almacenada como su código más 96.

Después del nombre siguen 2 bytes que contienen la longitud total (en bytes) que ocupa el resto del conjunto, es decir, 5 por cada elemento, más 2 por cada dimensión, más 1 para el byte que indica el número de dimensiones. A continuación viene el byte en el que está registrado el número de dimensiones. Seguidamente se utilizan 2 bytes para cada dimensión y, por último, 5 bytes por elemento (fig. 26).

Los elementos son registrados de la siguiente forma: primero todos aquellos cuyo primer subíndice es 1, después aquellos cuyo primer subíndice es 2, etc.

Cada grupo que tiene el mismo primer subíndice se ordena de la misma forma y así sucesivamente.

5.° *Variable de control*

La variable de control tiene por número una sola letra que se registra como su código más 192, como ya vimos en el Capítulo dedicado a los bucles FOR NEXT. En ella quedan también registrados el valor (5 bytes), el límite, el paso y la línea a la que tiene que saltar (Figura 27).

6.° *Conjunto de caracteres*

Un conjunto de caracteres es reconocido porque su nombre se registra como el código más 160. Está estructurado de forma parecida al conjunto de números, sólo que por cada elemento se utiliza un sólo byte.

Después del área de variables hay 1 byte que contiene el valor 80 H (hexadecimal) o lo que es lo mismo 1280.

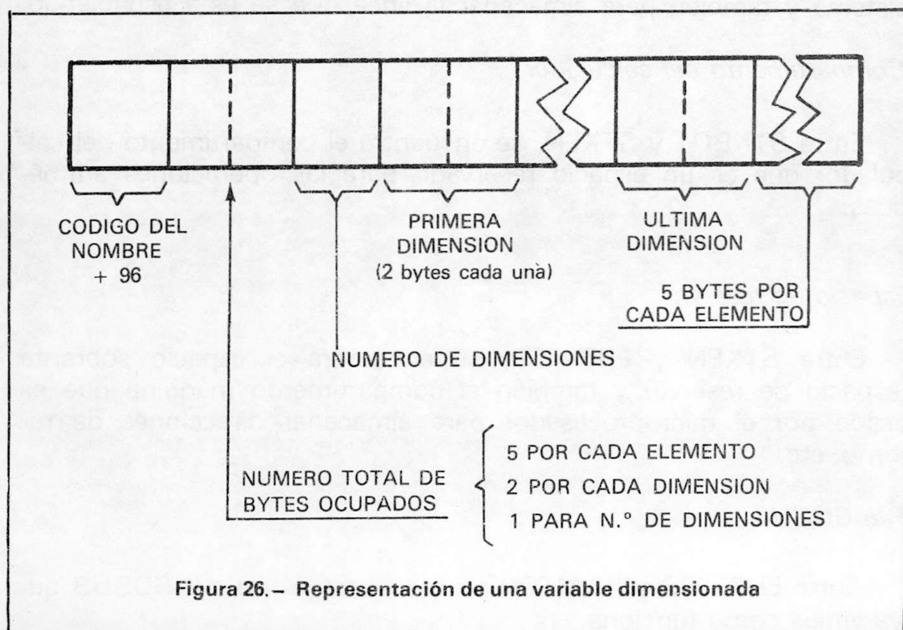


Figura 26.– Representación de una variable dimensionada

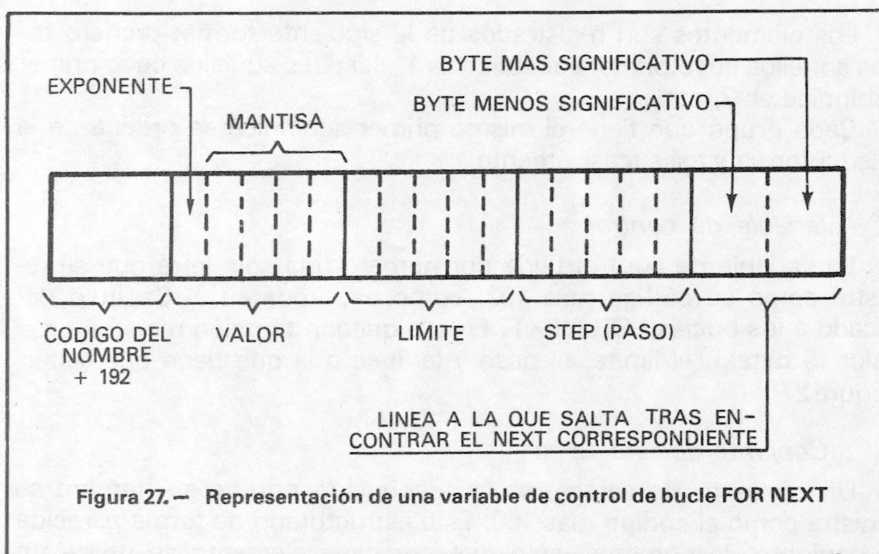


Figura 27. — Representación de una variable de control de bucle FOR NEXT

Espacio de trabajo

Entre E—LINE y STKBOT hay un espacio usado para trabajar el sistema y también para almacenar la línea que se está escribiendo.

Compartimento del calculador

Entre STKBOT y STKEN, se encuentra el compartimento del calculador que es un espacio reservado para las operaciones aritméticas.

Espacio de reserva

Entre STKEN y ERR—SP, se encuentra el espacio sobrante (espacio de reserva) y también el compartimento máquina que es usado por el microprocesador para almacenar direcciones de retorno, etc.

Pila GOSUB

Entre ERR—SP y RAMTOP se encuentra la pila del GOSUB que ya vimos cómo funciona.

La variable RAMTOP indica la dirección del último byte de memoria accesible por un programa BASIC. Si este byte no es el último, el espacio que queda entre él y el último byte de memoria es un buen lugar para almacenar rutinas en código máquina.

Hemos estudiado cómo está distribuida la memoria y se ha observado que la capacidad total es invariable (o se tiene 1 K ó 16 K, etc.). Las distintas zonas no son de longitud fija y las divisiones entre ellas se almacenan en las variables del sistema.

La zona de las variables del sistema es la única que tiene longitud fija, pues siempre son las mismas.

En el próximo Capítulo se examinará cómo resulta posible saber qué valores están contenidos en ellas o en cualquier otra dirección de memoria.

CAPITULO 15

ACCESO A LA MEMORIA. — PEEK Y POKE

Una vez conocida la distribución de la memoria es interesante poder obtener el valor del byte almacenado en una determinada posición y también poder cambiarlo en los casos en que ello resulte posible.

Se puede obtener el valor de cualquier dirección, pero no es posible cambiar el valor de los 8192 primeros bytes que configuran la ROM, ya que en ellos se encuentran almacenadas las instrucciones en código máquina que permiten que el ZX81 interprete el BASIC. Con posterioridad se comentará brevemente cómo se realiza la interpretación.

Debido a ello, cualquier intento de cambiar un byte de la ROM no tendrá efecto alguno.

Las variables del sistema también pueden cambiarse aunque esto es susceptible de proporcionar sorpresas desagradables en algunos casos.

La sentencia que nos permite obtener el contenido de una dirección de memoria es la función PEEK. Su formato es el siguiente:

PEEK dirección

La dirección es el argumento de la función, que debe estar en la serie 0—65535 (ya que todas las direcciones se almacenan en dos bytes).

El resultado es el valor del byte cuya dirección se indica en el argumento y por tanto es siempre un número entero en la serie 0—255.

Instruccion POKE

La instrucción POKE nos permite almacenar un valor en una direc-

ción de memoria, perdiéndose el que había previamente al igual que ocurre con las variables.

El formato de la instrucción POKE es:

POKE dirección, valor

La dirección debe estar en la serie 0—65535, pero si es menor que 8192 no tendrá ningún efecto, como se ha hecho observar anteriormente.

El valor debe ir separado de la dirección por una coma y tiene que estar en la serie 0—255 puesto que se almacena en un byte. Ejemplo:

POKE 17000, 54 almacena en el byte de dirección 17000 el número 54 perdiéndose por completo lo que había anteriormente en dicha corrección.
PRINT PEEK 17000 imprime en pantalla el valor del byte de la corrección 17000. Si hace esto inmediatamente después debe obtener 54.

Si la dirección estuviera entre 8192 y 16383, tampoco tendría efecto puesto que ya se vio que estas direcciones no existen físicamente, aunque si se hace PEEK de alguna de ellas, se obtiene un resultado. Pero esto es debido a que los valores que están en la serie 8192—16383, se cambian por valores entre 0—8192. Esto se consigue simplemente poniento el bit 14 a 0 (en realidad, lo que ocurre es que no está conectado).

Algo similar ocurre cuando la dirección es mayor que el máximo de memoria disponible. En caso de que se disponga de un módulo de 64 K la situación es distinta puesto que se utilizan también los 8 K siguientes a la ROM (direcciones 8192, 16383).

Obtención e introducción en la memoria de números enteros

1.º *Enteros de 1 byte*

El ZX81 utiliza enteros de 1 byte en muy contadas ocasiones, por ejemplo para indicar el número de dimensiones de un conjunto o para representar un carácter. Ya que los códigos de los caracteres van de 0 hasta 255, también se usan en algunas variables del sistema.

Las operaciones necesarias para obtenerlos o introducirlos, son obvias: supongamos que queremos obtener el valor de la dirección A. Haremos PEEK A. Si queremos introducir un valor V en la dirección A haremos POKE A, V.

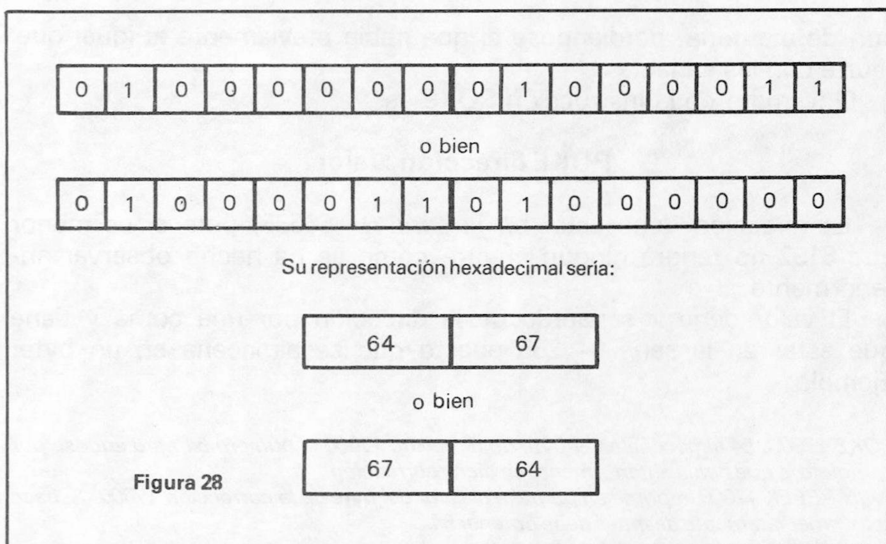


Figura 28

2.° *Enteros de 2 bytes*

Los enteros de 2 bytes son los más utilizados. Por ejemplo: números enteros de 2 bytes son la longitud total de las cadenas y conjuntos, el valor de las dimensiones, algunas variables del sistema y las direcciones incluso las que son menores que 255 (en cuyo caso uno de los bytes tiene el valor 0).

Para entender correctamente lo que sigue, convendrá repasar el Capítulo dedicado a las distintas formas de representación de números.

Está claro que en cualquier número almacenado en 2 bytes uno de los dos es más significativo que el otro. Por ejemplo, el número 16541 en representación decimal, si se calcula su representación binaria se obtiene: 0100000001000011.

El primer 0 se incluye para que ocupe exactamente 2 bytes. El ZX81, puede almacenar este número de dos formas distintas (fig. 28).

El primer caso sería más lógico ya que el byte más significativo es el primero, pero esto ocurre muy raramente en el ZX81 y en muchos casos los números se almacenan con el byte menos significativo en primer lugar, como ocurre en el caso de las variables del sistema que ocupan 2 bytes.

Por tanto, siempre que se desee obtener o introducir directamente en memoria un entero de 2 bytes, se debe conocer previamente en qué posición tiene el byte más significativo.

Todo ello se debe a que para obtener el número hay que multiplicar el byte más significativo por 256 y añadirle el valor del menos significativo.

Esto se ve claramente en el caso del número que se ha escogido como ejemplo. Examinando su representación binaria se observa que el primer byte tiene un valor de 64 mientras que el segundo tiene uno de 67. Realizando las operaciones descritas anteriormente obtendremos que el número es igual a 256 × 64 + 67 = 16541. En cambio, si suponemos que el byte más significativo es el segundo, el número que se obtiene es 64 + 256 × 67 = 17216.

Obtención PEEK

Resumiendo, para obtener un número entero de 2 bytes almacenado en las direcciones A y A + 1 se actúa del siguiente modo:

1.° Si el byte más significativo es el primero:

n.° = 256 × PEEK A + PEEK (A + 1)

2.° Si el byte más significativo es el segundo:

n.° = PEEK A + 256 × PEEK (A + 1)

Introducción POKE

Para introducir un entero de 2 bytes en memoria lo primero que hay que hacer, es descomponer el número en los valores que se han de almacenar en cada byte. Veámoslo con un ejemplo:

Para obtener los valores de los 2 bytes que representan al número 16541, se divide por 256 y el cociente es el byte más significativo mientras que el resto es el menos significativo.

En general, si queremos introducir un número V en memoria, de forma que el byte más significativo esté en la dirección A y el menos significativo esté en la dirección A + 1, se efectuará lo siguiente:

1.° POKE A, INT (N/256)
2.° POKE A + 1, N—INT (N/256) * 256

Observe el factor que va detrás de la coma de la segunda instrucción y verá que es el resto de dividir el número N por 256. Asimismo, realice esta operación dando valores correctos a N y a A; es decir, que tanto N como A estén en la serie 0-65535.

Una vez hecho esto, aplique los resultados del apartado anterior y verá cómo recupera de nuevo el número N.

Para efectuar pruebas con PEEK y POKE, (se procurará hacerlo

en una área de memoria, donde no afecte a otra información que tenga almacenada.

Si al principio no funciona, no hay que apurarse; ya se verán más adelante sitios seguros en que pueda hacerse.

Escriba el siguiente programa:

```
10 FOR I=0 TO 50
20 PRINT PEEK (16509+I);TAB 8
30 PRINT CHR$ PEEK (16509+I)
40 NEXT I
```

Ejecútelo con RUN y obtendrá:

```
0
10        ▒
20        =
0
235       FOR
46        I
20        =
28        0
126       ?
0
0
0
0
0
223       TO
33        5
28        0
126       ?
134       ▖
72        ?
0
0
```

Este programa proporciona los 44 primeros bytes del área de programa, indicando para cada uno de ellos, dirección de memoria y carácter al que representa dicho valor.

El hecho de que se imprima también el carácter es útil en algunas ocasiones para observar las palabras clave sin necesidad de tener que descifrar el código.

Vamos a comentar los primeros 22 números que son los que corresponden a la primera pantalla.

Los dos primeros representan el primer número de línea; los dos

siguientes (2 y 0) representan la longitud de línea. Observe que el byte más significativo es el segundo.

La representación de los tres siguientes es clara por la traducción del código.

Luego sigue la representación del 0 del siguiente modo: primero como un carácter normal, es decir, mediante su código que es 28; después sigue el valor 126, que es el que precede siempre a las constantes reales y luego la representación del 0 en 5 bytes como un número real.

Posteriormente, la sentencia TO y el número 50 representado de igual manera que el 0; es decir, primero su carácter, después el 126 y la forma real en coma flotante 134, 72, 0, 0,0.

Por último, el valor 118 que es el código de NEW LINE y nos indica el final de la línea.

Para continuar con el listado, pulse CONT. Si ha entendido esto correctamente, pruebe a hacerlo con otros programas, añadiendo éste al final separado por una instrucción STOP. Si lo cree necesario, luego haga RUN desde la línea en que empieza este programa y le saldrá en pantalla la codificación de programa que usted ha introducido en primer lugar.

Se hará uso de la instrucción POKE, por ejemplo, para cambiar el primer número de línea: haga POKE 16510, 80, NEW LINE y luego pulse LIST; observará que la primera línea tiene ahora el número 80. Si quiere dejarlo como estaba, haga POKE 16510, 10.

Si se quiere cambiar la variable de control, debe hacerse en todos los sitios en que aparezca. Por ejemplo, POKE 16514, 38, NEW LINE y LIST, como 38 es el código de A, la variable de control que antes era I ahora es una A.

El programa no funcionará hasta que no cambie la variable de control en todas partes.

CAPITULO 16

VARIABLES DEL SISTEMA

Las variables del sistema son utilizadas por el ZX81 para controlar los programas en BASIC. Algunas de ellas contienen información de la que puede hacer uso un programa en BASIC, por ejemplo, para saber dónde empiezan o terminan las distintas zonas de la memoria en un momento determinado o reservar espacio seguro para los programas en código máquina.

Los nombres que reciben son únicamente nemotécnicos para recordar la función que tienen. Por lo tanto, no podemos saber su valor llamándolas por su nombre ni tampoco cambiarlo, asignándolo como una variable normal. El único modo de obtener el valor que contienen es mediante la función PEEK, puesto que las direcciones en que se encuentran son conocidas y se explicarán a lo largo del capítulo. Asimismo, el único modo de cambiar el valor de una de las variables es mediante la instrucción POKE.

A continuación se comentan una por una todas las variedades del sistema indicando: Número de bytes de que constan, su dirección, el nombre nemotécnico y alguno de sus posibles usos.

Recuerde que para las variables de dos bytes, el segundo de ellos es el más significativo y el primero es el menos significativo.

16384 ERR NR: Esta variable ocupa 2 bytes y en ella es donde se registran los códigos de error menos una unidad; por lo tanto cuando no hay ningún error debe dar 255, ya que cuando se interpreta el valor de 1 byte con valores negativos desde el 255 hasta el 128 representan a los números —1, —2, —3, ... —128 y desde el 0 hasta el 127 se representan los números positivos. A este tipo de notación se le llama complemento a 2. Esta variable se puede utilizar para

forzar detenciones por error creando nuevos códigos: menos los números del 128 al 255 que no tienen efecto pues se consideran negativos.

—Del 0 al 14 proporcionan los códigos de error ya conocidos (1... ...F).

—Del 15 al 34 inclusive proporcionan como código de error las restantes letras del alfabeto.

—Desde el 35 al 98 se detiene el programa, pero no proporciona ningún código de error y puede desordenar el archivo de imagen.

—Del 99 al 127 proporciona como código de error los caracteres cuyos códigos van desde 128 al 156. Para ver cómo funciona, escriba el siguiente programa:

```
10 INPUT A
20 POKE 16384,A
```

Pruebe a dar diferentes valores a la variable A y observe cómo se detiene el programa con diferentes códigos de error.

16385 FLAGS: Esta variable ocupa un byte y contiene varias banderas que controlan el sistema BASIC. Una bandera es algo que puede tener dos posiciones: «subidà o bajada» y corresponde a un bit cada una. No es aconsejable hacer pruebas con esta variable pues puede destruirse toda la información. Algunas de las banderas que contiene son:

—Bit 0: Supresión de espacios previos.
—Bit 1: Controla la impresora.
—Bit 2: Selecciona entre el modo K o L, o bien F o G.
—Bit 7: Está a 0 durante la comprobación de sintaxis.

16386 ERR—SP: Esta variable ocupa 2 bytes y contiene la dirección del comienzo de la pila de GOSUB.

16388 RAMTOP: Esta variable ocupa 2 bytes y contiene la dirección del primer byte inexistente de memoria. Si no se cambia, su valor depende de la memoria de que se disponga.

Esta variable es muy útil puesto que si se rebaja su valor y se ejecuta seguidamente NEW (esto ha de hacerse antes de introducir cualquier programa), provoca que el espacio que queda desde la dirección contenida en RAMTOP hasta el final de la memoria no sea

afectado ni por NEW, ni por RUN, ni por CLEAR e incluso se conserva lo que hay en él cuando se carga un programa. Este espacio es muy útil para tener rutinas en código máquina de las que hablaremos brevemente en el capítulo siguiente.

En definitiva, el espacio que queda es como si no existiera para el sistema BASIC.

Haga POKE 1638, 175 y POKE 16389, 65 NEW e intente introducir un programa y observará que al poco tiempo el ZX81 actúa como si se le hubiese terminado la memoria, puesto que para él la última dirección de memoria es la 16640; es decir, le queda 1/4 de la RAM pero la mitad está ocupado por las variables del sistema.

16390 MODE: Esta variable ocupa un byte y en ella está contenido un código que especifica el modo en que se encuentra el cursor. Para ver cómo actúa escriba el siguiente programa:

```
10 INPUT A
15 PRINT PEEK 16390
20 POKE 16390,A
35 PRINT PEEK 16390
40 GOTO 10
```

16391 PPC: Esta variable ocupa 2 bytes y contiene el número de línea de la sentencia que se está ejecutando. Haga un programa en el que todas las líneas sean:

PRINT: PEEK 16391 + 256 * PEEK 16392 y se irán escribiendo los números de las líneas por los que pasa.

16393 VERSN: Esta variable ocupa 1 byte y es la primera dirección de memoria que se graba. Esta variable y todas las que siguen a continuación son conservadas por SAVE, junto con el programa.

16394 E—PPC: Esta variable ocupa 2 bytes e indica el número de la línea que tiene el cursor de programa. Se puede utilizar para situarlo donde nos plazca del mismo modo que lo hace LLIST.

16396 D—FILE: Esta variable ocupa dos bytes y contiene la dirección en que empieza el archivo de imagen. No es aconsejable cambiarle el valor, pero es muy útil si se quieren escribir caracteres en pantalla «Pokeándolos» directamente en el archivo de imagen. Escriba

el siguiente programa y observará cómo se imprime en la posición que usted introduzca el carácter cuyo código se pide:

```
   10 INPUT POSICION
   20 INPUT CODIGO
   30 POKE PEEK 16396+256*PEEK 16
397+POSICION,CODIGO
   40 GOTO 10
```

Procure introducir un código cuyo carácter no ocupe más de una posición de pantalla pues si no se bloquea el programa.

16398 DF—CC: Esta variable ocupa 2 bytes e indica en qué dirección del archivo de imagen se va a imprimir el próximo carácter.

Si la pantalla está vacía contiene el mismo valor que D—FILE, pero va aumentando a medida que se va llenando la pantalla. Con este sencillo programa se puede observar cómo varía.

```
   10 PRINT PEEK 16398+256*PEEK 1
6399
   20 PRINT "A",
   30 GOTO 10
```

Haga correr el programa y observará que cada vez el valor de DF—CC aumenta en 33 ya que la letra A se imprime una vez en cada línea. Como se vio en el Capítulo anterior, cada línea consta de 33 caracteres (32 + 1 NEW LINE).

16400 VARS: Esta variable ocupa 2 bytes y contiene la dirección del primer byte del área de variables.

16402 DEST: Esta variable ocupa 2 bytes y contiene la dirección de la variable que se está asignando en aquel momento.

16404 E—LINE: Esta variable ocupa 2 bytes y contiene la dirección del primer byte del área de trabajo.

16406 CH—AD: Esta variable ocupa 2 bytes y contiene la dirección en el área de programa del carácter que sigue al argumento de PEEK

o del NEW LINE al final de una sentencia POKE. Es muy improbable que le sirva para algo. Para ver cómo funciona escriba el siguiente programa:

```
10 PRINT CHR$ PEEK (PEEK 16406
+256*PEEK 16407)
```

Cuando haga RUN debe aparecer el símbolo +, que es el que sigue al argumento de PEEK en el momento de la ejecución de la línea.

16408 X—PTR: Esta variable ocupa 2 bytes y contiene la dirección del carácter que precede al indicador S . Cuando no hay ningún error de sintaxis contiene un 0.

16410 STKBOT: Esta variable ocupa 2 bytes y contiene la dirección del primer byte del compartimento del calculador.

16412 STKENT: Esta variable ocupa 2 bytes y contiene la dirección del último byte del calculador.

16414 BERG: Esta variable ocupa 1 byte y es un registro usado para operaciones aritméticas.

16415MEM: Esta variable ocupa 2 bytes y contiene la dirección de lugar donde se guardan los números en coma flotante con los que hay que operar. Generalmente remite a MEMBOT excepto en algunos casos, por ejemplo cuando MEMBOT está lleno.

16417 No usada: Puede servir para almacenar números entre 0 y 255.

16418 DF—SZ: Esta variable ocupa 1 byte y contiene el número de líneas que hay en la parte inferior de la pantalla. Puede modificarse desde el interior de un programa para obtener 24 líneas de pantalla disponibles (poniendo un 0). Cuando el programa termine, volverá a tener un 2 que es el número normal de líneas. También se puede utilizar para conseguir que SCROLL afecte sólo a un número predeterminado de líneas de la parte superior, como se muestra en el siguiente programa:

```
  10 INPUT A
  20 POKE 16418,A
  30 SCROLL
  40 PRINT " SCROLL DE ";24-A;"
LINEAS"
  50 GOTO 30
```

La variable A contiene el número de líneas de la parte baja de la pantalla.

16419 S—TOP: Esta variable ocupa 2 bytes y contiene el número de la primera línea que se lista al pulsar NEW LINE.

16421 LAST—K: Esta variable ocupa 2 bytes y contiene un código referido a la tecla que se está pulsando en aquel momento. En esta variable podemos considerar los dos bytes por separado:

El número contenido en el primer byte, depende de la zona horizontal en que esté la tecla que se está pulsando (fig. 29).

Si no se pulsa ninguna tecla, el primer byte (16427) contiene el valor 255.

Si se pulsa una tecla de la zona 1 contiene 254.
Si se pulsa una tecla de la zona 2 contiene 253.
Si se pulsa una tecla de la zona 3 contiene 251.
Si se pulsa una tecla de la zona 4 contiene 247.
Si se pulsa una tecla de la zona 5 contiene 239.
Si se pulsa una tecla de la zona 6 contiene 223.
Si se pulsa una tecla de la zona 7 contiene 191.
Si se pulsa una tecla de la zona 8 contiene 127.

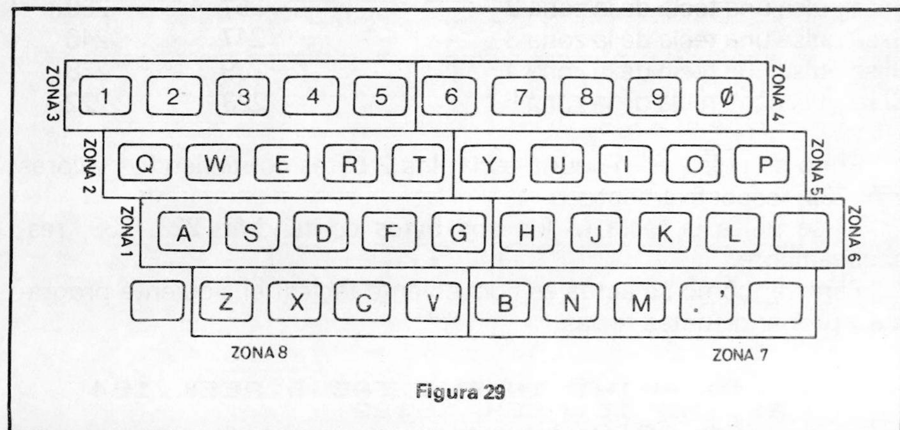


Figura 29

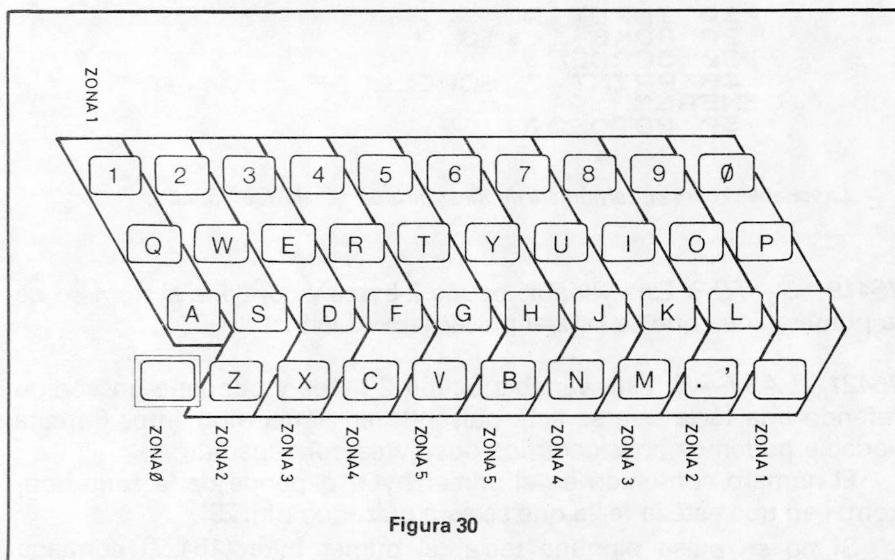


Figura 30

Para los valores del 2.º byte se divide el teclado en zonas verticales (figura 30).

El valor registrado en el 2.º byte es distinto si se pulsa SHIFT.

	con SHIFT	sin SHIFT
Si no se pulsa ninguna tecla, el segundo byte contiene:	254	255
Si se pulsa una tecla de la zona 1	253	252
Si se pulsa una tecla de la zona 2	251	250
Si se pulsa una tecla de la zona 3	247	246
Si se pulsa una tecla de la zona 4	239	238
Si se pulsa una tecla de la zona 5	223	222

Si no se pulsa así ninguna tecla, los 2 bytes contienen los valores 255, 255, respectivamente.

Si se pulsa la tecla M los dos bytes contendrán 223, 239, respectivamente.

Para ver cómo se actúa es conveniente teclear el siguiente programa y probar distintas teclas.

```
   10 PRINT INKEY$;TAB 8;PEEK 164
21;TAB 15;PEEK 16422
   20 GOTO 10
```

Es obvio que ésta es la variable usada por INKEY$, para reconocer qué tecla es pulsada en un momento determinado.

16423: Esta variable ocupa un byte. Cuando el teclado no está en situación de admitir una entrada contiene el valor 255.

16424 MARGIN: Esta variable ocupa un byte y contiene el número de líneas en blanco por encima o por debajo de la pantalla. 55 en Europa, 31 en América.

16425 NXTLIN: Esta variable ocupa 2 bytes y contiene la dirección en el área de programa de la próxima línea a ejecutar.

16427 OLDPPC: Esta variable ocupa 2 bytes y contiene el número de la última línea ejecutada. Se utiliza para diversos propósitos: por ejemplo, para que CONT sepa dónde tiene que saltar.

16429 FLAGX: Esta variable ocupa 1 byte y contiene varias banderas:
Bit 0, indica si la variable que se trata es o no dimensionada.
Bit 1, indica si la variable que se trata está definida o no.
Bit 5, está a 1 durante la espera de INPUT.
Bit 6, está a 1 si la variable de INPUT es numérica.

16430 STRLEN: Esta variable contiene 2 bytes y contiene la longitud de la última cadena asignada.
16432 T—ADDR: Esta variable ocupa 2 bytes e indica la dirección en la ROM de las tablas de parámetros para la comprobación de la sintaxis. Uno de los bits se emplea para distinguir entre PLOT y UNPLOT.

16434 SEED: Esta variable ocupa 2 bytes y es la raíz para RND, es decir, donde se almacena el valor dado después de RAND. Puede tener una utilidad suplementaria que no tiene nada que ver con su función, que consiste en evitar tener que hacer los cálculos del resto y el cociente de la división por 256, cuando se utiliza la instrucción POKE en una variable de 2 bytes. Esto se consigue usándola del siguiente modo: supongamos que queremos introducir un entero de 2 bytes N en las direcciones D y D + 1. En lugar de hacer POKE D, INT (N/256) y POKE D + 1, N — INT (N/256) * 256 se puede hacer:

```
RAND N
POKE D + 1, PEEK 16434
POKE D, PEEK 16435.
```

O bien intercambiando D y D + 1, si queremos que el byte más significativo sea el de la dirección D + 1.

16436 FRAMES: Esta variable ocupa 2 bytes y va variando a razón de 50 unidades por segundo. El bit 15 siempre es un 1 y los bits del 0 al 14 van disminuyendo a esta velocidad. Es la variable usada por PAUSE que actúa del siguiente modo: pone a 0 el bit 15 e introduce en los bits del 0 al 14 la duración de la Pausa. Cuando ésta llega al valor 0, la pausa termina. Si la pausa es terminada porque se ha pulsado una tecla, el bit 15 pasa nuevamente a tener el valor 1. Si no se utiliza PAUSE, como que el bit 15 siempre está a 1 cuando llega a 32768 (1000000000000000) vuelve a empezar desde 65535.

Si quiere observar cómo cambia FRAMES utilice este programa:

```
10 PRINT PEEK 16436+256*PEEK 1
6437;" ";
20 GOTO 10
```

16438 COORDS 1: Esta variable ocupa 1 byte y contiene la coordenada X del último punto que se ha utilizado en una instrucción PLOT o UNPLOT.

16439 COORDS 2: Esta variable ocupa 2 bytes y contiene la coordenada Y del último punto que se ha utilizado en una instrucción PLOT o UNPLOT.

16440 PR—CC: Esta variable ocupa 1 byte y contiene la posición en el buffer de la impresora (PRBUFF), del próximo carácter que se va a imprimir.

16441 S—POSN 1: Esta variable ocupa 1 byte y contiene el número de columna de la posición de PRINT, empezando a contar desde la derecha.

16442 S—POSN 2: Esta variable ocupa 1 byte e indica el número de línea de la posición de PRINT empezando a contar desde abajo.

Estas dos variables son las usadas por PRINT AT.

16443 CDFLAG: Esta variable ocupa 1 byte y contiene varias banderas:

Bit 6: si está a 0 indica modo FAST.
si está a 1 indica modo SLOW.

Bit 7: es una copia del bit 6.

Normalmente en modo FAST contiene un 0 (00000000) y en modo SLOW contiene 192 (11000000).

16444 PRBUFF: Esta variable contiene 33 bytes y es el buffer de la impresora, es decir, la memoria intermedia de la próxima línea que se va a imprimir; por lo tanto, el trigésimotercero carácter es un NEW LINE. Si se carece de impresora, estos 33 primeros bytes se pueden utilizar para almacenar números mediante POKE, pero hay que tener en cuenta que si se usa alguna función de impresora se pierde todo lo almacenado en el buffer.

16477 MEMBOT: Esta variable ocupa 30 bytes y es una memoria para el calculador que da cabida a 6 números en coma flotante.

16507: Esta variable ocupa 2 bytes. Si no se emplea, se puede utilizar para almacenar un número entero en la serie 65535.

CAPITULO 17

INTRODUCCION A CODIGO MAQUINA

No es el objeto de este libro enseñar a programar en código máquina pero se intentará dar una visión general de su uso en el ZX81, incluyéndose las instrucciones con sus códigos para aquellos que ya conocen el lenguaje del microprocesador Z80.

Ya se explicó en la Introducción en qué consiste el código máquina. Sus ventajas respecto al BASIC son dos:

1.° Un programa en código máquina emplea mucha menos memoria que otro programa equivalente, escrito en cualquier lenguaje de alto nivel, en nuestro caso el BASIC.

2.° Los programas en código máquina se ejecutan más rápidamente. Esta rapidez es muy importante en juegos gráficos o en programas en los que el usuario espera respuesta constantemente. El tiempo ganado es el que corresponde al que pierde el intérprete traduciendo cada instrucción del lenguaje de alto nivel al lenguaje máquina.

La principal desventaja radica en la dificultad y laboriosidad de su escritura, pues sus instrucciones son simplemente números en hexadecimal.

Otra gran diferencia que existe entre BASIC y código máquina es que este último carece de variables en el sentido del BASIC. En su lugar se utilizan unas zonas llamadas registros, cada uno de los cuales tiene una capacidad de un byte; es decir, sólo pueden almacenar números entre 0 y 255. Los nombres de estos registros se encuentran en la figura 31.

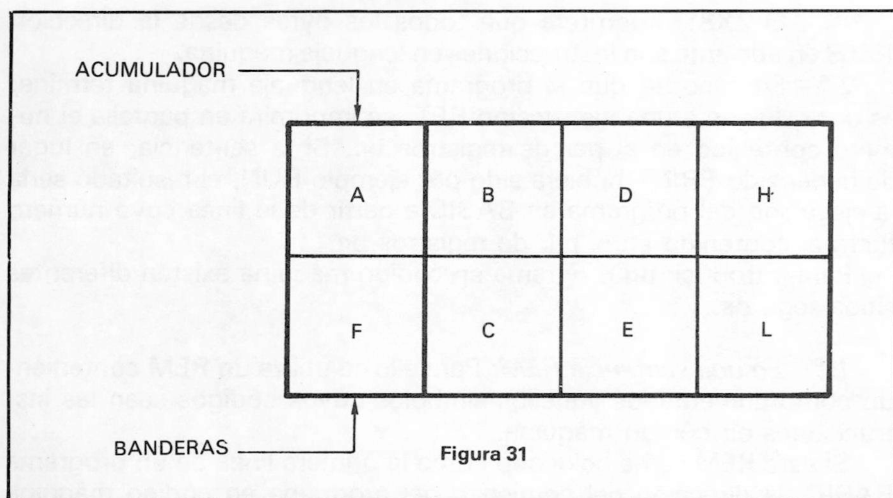


Figura 31

Algunos de ellos se agrupan en pares para poder almacenar números más grandes, por ejemplo bc, de y hl.

Existe otro juego de registros alternativos para cada uno de los anteriores y también otros que tienen una función más específica.

En el ZX81, el código máquina hay que situarlo en un lugar seguro, para que no sea sobreescrito por un programa en BASIC, una variable, etc.

Las instrucciones reciben unos nombres llamados mnemotécnicos, que sirven para recordar lo que realizan cada una, pero que no son entendidos por el microprocesador Z80.

Es muy importante conocer exactamente la dirección de memoria en que empieza el programa en código máquina, puesto que para ejecutarlo se debe utilizar la función USR, seguida de la dirección de inicio. La función USR tiene el siguiente formato:

Sentencia USR dirección.

La sentencia puede ser cualquier instrucción del BASIC. Las más utilizadas son PRINT, RUN y RAND. USR actúa del siguiente modo: ejecuta a partir de la dirección que se indica, el programa en código máquina, hasta que encuentra una instrucción RET cuyo código es 201, en que vuelve al BASIC y con el valor contenido en el par de registros bc del Z80, ejecuta la sentencia que se ha dado previamente. Por ejemplo: PRINT USR 16509 actuaría de la siguiente forma:

1°. El ZX81, interpreta que todos los bytes desde la dirección 16509 en adelante son instrucciones en lenguaje máquina.

2.° En caso de que el programa en lenguaje máquina termine, es decir, llegue a una instrucción RET, se imprimirá en pantalla el número contenido en el par de registros bc. Si la sentencia, en lugar de haber sido PRINT hubiera sido por ejemplo RUN, el resultado sería la ejecución del programa en BASIC a partir de la línea cuyo número fuera el contenido en el par de registros bc.

Para introducir un programa en código máquina existen diferentes sitios seguros:

1.° En una sentencia REM: Para ello se utiliza un REM conteniendo como comentarios aquellos símbolos cuyos códigos sean las instrucciones en código máquina.

Si este REM se ha colocado como la primera línea de un programa BASIC, la dirección del comienzo del programa en código máquina será 16514. Puesto que hay que tener en cuenta que primero van dos bytes que indican el número de línea (16509 y 16510), luego siguen los dos bytes que indican la longitud total de la línea (16511 y 16512), a continuación en la dirección 16513, está el código de REM y a partir de 16514 en adelante vienen los comentarios que serán tomados como instrucciones en código máquina.

Si el REM no se coloca como primera línea habrá que calcular con toda exactitud la dirección en que empieza el comentario propiamente dicho. Como ejemplo, puede escribirse el siguiente programa:

```
10 REM ▘▙ Y▒LN ▒▒"4 PLOT TAN
```

o bien éste,

```
10 REM ▘▙ Y□LN ▒▒"4 PLOT TAN
```

colocando en el lugar marcado con □ cualquier carácter del teclado que ocupe una única posición en pantalla. Para que no haya duda sobre qué símbolos son los que hay que escribir se muestran aquí sus códigos, puesto que con un solo error, el programa no funcionaría:

```
▘        1
▙        188
,        0
Y        62
□        □
```

```
LN        205
▒         5
▒         6
"         11
4         32
 PLOT     246
TAN       201
```

Para escribir PLOT escriba primero THEN y luego bórrelo. No deletree ninguna de las sentencias que aparecen pues no tendrían el mismo código y el programa no funcionaría.

Este programa se ejecuta mediante PRINT USR 16514 ó RAND USR 16514 o GOTO, etc... e imprime a gran velocidad el carácter que se ha situado en la posición marcada.

Un programa en código máquina, almacenado en REM, se puede grabar sin ningún tipo de problemas.

2.º En el área de variables: Otro lugar en que se puede almacenar un programa en código máquina es el área de variables. Para ello hay que reservar espacio en una cadena dimensionada, con tantos bytes como tenga el programa.

La dirección de inicio se calcula mediante la siguiente expresión: PEEK 16400 + 256* PEEK 16401, suponiendo que la sentencia DIM se haya hecho antes que cualquier otra asignación de variable. Pruebe a introducir en una variable de cadena lo mismo que en el comentario del REM del apartado anterior y ejecute el programa con la nueva dirección de inicio.

Se tendrá en cuenta que si se usa RUN o CLEAR se borran las variables y por tanto el programa entero en código máquina.

3.º Después de RAMTOP: Si se cambia el valor de la variable RAMTOP mediante la sentencia POKE y NEW, queda reservado un espacio en las últimas direcciones de memoria según sea el valor introducido. En este espacio se pueden escribir programas en código máquina «Pokeando» los códigos de las instrucciones o más cómodamente, utilizando el siguiente programa.

```
 5 PRINT "DIRECCION INICIAL: "
10 INPUT X
13 PRINT X
20 LET A$=""
```

```
  30 IF A$="" THEN INPUT A$
  40 IF A$="S" THEN STOP
  50 POKE X,16*CODE A$+CODE A$(2
) -476
  60 LET X=X+1
  70 LET A$=A$(3 TO )
  80 GOTO 30
```

En primer lugar pide la dirección en la que va a empezar a almacenar el código máquina. El programa está preparado para recibir los códigos de instrucciones en hexadecimal. Pruebe a introducir los códigos del programa anterior todos seguidos en cuanto pida la cadena A$, pero tenga en cuenta que todos deben ser de dos digitos hexadecimales; es decir, 11 se escribirá como 0B, 1 como 01, 201 como C9, etc...

Cuando haya terminado introduzca «$» y el programa se detiene debido a la línea 40.

Ejecute el programa poniendo como argumento de USR la dirección de inicio que Vd. ha introducido al principio.

El hecho de que se reste 476 a los que se «pokea» es debido a que los números como carácter tienen el código aumentado en 28; es decir, el código de 0 es 28, el de 1 es 29, el de A (que en hexadecimal tiene el valor 10), es 38, etc. Entonces se resta 16 × 28 + 28 = 476, que es lo que se pone de más.

CAPITULO 18

AHORRO DE MEMORIA

La memoria del ZX81 es limitada: en su configuración standard, se dispone de un K (1024 bytes) de memoria para el usuario, pero que en realidad son menos, puesto que las variables del sistema ocupan 125 bytes con lo que el espacio queda reducido a 899 bytes. De ellos, por lo menos 25 y en el peor caso hasta 723, son usados por el archivo de imagen. También el compartimento del calculador, la pila de GOSUB y el compartimento máquina, gastan una cantidad variable de memoria. El objeto de este Capítulo es comentar algunas maneras de ahorrar memoria, que serán útiles sobre todo para aquéllos que no dispongan de ningún módulo de ampliación.

Si se dispone de 1 K no es difícil llenar la memoria. Veámoslo con un ejemplo: desconecte, si lo posee, el módulo de ampliación y escriba DIM A$(780), e intente introducir un programa. En seguida observará que la línea que está escribiendo se sube hacia arriba sin haber pulsado NEW LINE. Si continúa pulsando teclas, verá que en lugar de escribirse el carácter que intenta introducir, se borra uno de los que ya están escritos. Esto se debe a que el ZX81 está reduciendo el archivo de imagen para dar cabida a la línea que se está introduciendo. Aunque la línea aún existe en el área de programa, si se continúa escribiendo se puede perder completamente. En cualquier caso, el error 4, indica que no hay suficiente memoria para el proceso que se está realizando.

Si quiere comprobar lo que ocurre cuando la memoria está llena, disponiendo de un módulo, dimensione una cadena de aproximadamente 1000 bytes menos que el total de memoria disponible. Cuando

se encuentre que un programa no le cabe en la memoria, intente reducir el espacio ocupado mediante uno de los siguientes métodos:

1.° Repase el algoritmo; seguramente encontrará otro que haga lo mismo con menos operaciones.

2.° Reduzca al máximo el archivo de imagen utilizando frecuentemente CLS (esto no tiene efecto si se dispone de más de 4 K de RAM).

3.° Las constantes numéricas es una de las cosas que ocupan más memoria: 1 byte por cada carácter más 5 bytes por representación binaria más otro byte conteniendo 126, para separar dicha representación del resto del programa. Por lo tanto, si una constante tiene que ser usada varias veces, es mejor asignarla a una variable. Si esto último se realiza además como una orden y no como una línea de programa, se ahorrará mucha memoria, pero con el inconveniente de que es más difícil corregir el programa en caso de que haya algún error.

Ejemplo: La línea 10 IF S < 48 THEN LET A = 48 ocupa 34 bytes entre programa y variable, mientras que LET K=48 (sin número de línea) y 10 IF S < K THEN LET A=K ocupa 26 bytes.

4.° La función CODE también permite ahorrar memoria cuando hay que introducir números en la serie 0—255; así, por ejemplo: *LET S=CODE "A"*, ocupa 4 bytes menos que *LET S=38* y tiene el mismo efecto aunque se ejecuta más lentamente.

Para números mayores que 255 se puede usar VAL; por ejemplo, *LET A=VAL "52"* ocupa 3 bytes menos que *LET A="52"*.

Para el número 0 se puede utilizar NOT PI que ocupa sólo 2 bytes y para el número 1 PI/PI, que ocupa 3 bytes.

5.° Es conveniente utilizar una variable que no se necesite antes que crear una nueva puesto que la otra sigue ocupando memoria. Esto hay que tenerlo muy en cuenta en las variables de control de bucles FOR NEXT, ya que ocupan 18 bytes.

6.° Reducir los nombres de variables a una sola letra, así como suprimir todas las líneas de comentarios REM.

Asimismo haga que el programa termine en la última línea para evitar el empleo de la necesaria instrucción STOP.

7.° El uso adecuado de operaciones lógicas puede reducir ligeramente el gasto de memoria. Así, por ejemplo, 10 IF A=34 THEN PRINT "FIN" ocupa 23 BYTES, MIENTRAS QUE 10 PRINT "FIN" AND A=CODE " ", ocupa 18.

Recuerde también que una sentencia PRINT larga acostumbra a gastar menos que dos cortas.

Para saber cuánta memoria RAM ha utilizado, repase el Capítulo

dedicado a las variables del sistema y recuerde que: "D—FILE"—16509 indica lo que ocupa el programa.

(E—LINE)--VARS indica lo que ocupan las variables.

Sumando estos dos valores, se obtiene lo que ha gastado el programa aunque piense que después de E—FILE no todo lo que viene es espacio libre sino que el ZX81 ocupa algunos bytes para distintos fines. Además, se pueden tener rutinas en código máquina si ha cambiado el valor de RAMTOP.

Los módulos de ampliación de memoria

La marca Sinclair únicamente ofrece ampliación de 16 K pero existen en el mercado ampliaciones de 16, 32 y 64 K, de otras marcas.

El funcionamiento es el mismo para todas y la única diferencia estriba en la estética y las conexiones.

El módulo de 16 K, se diferencia en cuanto a estructura de la configuración en 1 K en que el archivo de imagen tiene una longitud fija de 793 bytes. Ocupa desde la dirección 16383 hasta 32767.

El módulo de 32 K, simplemente se diferencia en que añade al de 16 K la memoria correspondiente a las direcciones 32768 a 49151.

El módulo de 64 K añade al de 32 K la memoria correspondiente a las direcciones 49152 a 65535 y además ocupa las direcciones de las 8 K siguientes a la ROM que no son utilizadas en ninguna de las demás configuraciones. Estas 8 K sólo pueden ser utilizadas como almacén de datos entre programas o bien para rutinas en código máquina, pues no pueden ser accedidas por el BASIC.

CAPITULO 19

COMPARACION CON OTROS BASICS

El lenguaje BASIC ha evolucionado mucho desde su creación. Actualmente existen algunas versiones muy avanzadas que permiten una gran versatilidad de aplicaciones tanto científicas como de gestión.

En este Capítulo se pretende comparar el BASIC del ZX81 con otros más avanzados y, al mismo tiempo, intentar suplir de algún modo la falta de algunas instrucciones lo cual puede ser muy útil para ejecutar con el ZX81 programas realizados por otros ordenadores.

El modo de substituir estas instrucciones depende en gran manera del contexto del programa mismo. Debido a esto se darán únicamente ejemplos útiles para situaciones determinadas pero que el lector podrá modificar adecuándolas a su programa.

Muchos BASICS permiten líneas de programas con diferentes sentencias separadas por un signo separador (normalmente se trata de / o :), mientras que en el ZX81 hay que escribir cada sentencia en una línea diferente.

La instrucción RENUMBER que permite renumerar a un programa a partir de un número de línea y de un intervalo dados, se puede sustituir por el siguiente programa optimizado, añadiéndolo al final del programa que se quiera renumerar, separado por una sentencia STOP.

```
9984 STOP
9985 PRINT "ENTRE LA PRIMERA LIN
EA Y LUEGO EL INTERVALO "
9986 INPUT A
9987 INPUT B
9988 LET I=A
9989 LET L=16508
9991 IF PEEK L=116 THEN GOTO 999
4
9992 LET L=L+1
```

```
9993 GOTO 9991
9994 IF PEEK (L+1) >=39 THEN STOP

9995 POKE (L+1),INT (I/256)
9996 POKE (L+2),I-INT (I/256)*25
6
9997 LET I=I+6
9998 LET L=L+3
9999 GOTO 9991
```

Para utilizar este programa, debe introducirse antes de teclear el que se quiere numerar. Se puede utilizar con cualquier módulo de memoria, pero si se usa disponiendo únicamente de 1 K, el espacio que queda para su programa es muy pequeño.

El programa actúa buscando los códigos de NEW LINE a través de toda el área de programa, ya que NEW LINE es el separador de dos líneas de programa. Esto se realiza en un bucle de 3 líneas (9991, 9992 y 9993) y cuando se encuentra con el valor 118 (código de NEW LINE) salta a la línea 9994 donde primero se realiza un «test» para que no se renumere el mismo y luego introduce en los dos bytes siguientes al que se ha encontrado 118 el nuevo número de línea.

Para que también renumere la primera línea se debe escribir previamente POKE 16508, 118 puesto que antes del primer número de línea no hay ningún NEW LINE.

Se ejecutará con RUN 9984; y habrá que tener en cuenta que una vez terminada la renumeración se deben cambiar los números de línea a los que saltan GOTO y GOSUB, pues no son cambiados por el programa, aunque usted pueda variarlo para que reconozca los GOTOs y GOSUBs de la misma manera que reconoce NEW LINE, y poder cambiar así los números de línea.

Para conseguir algo similar a la instrucción TRACE, que permite ejecutar un programa línea a línea, no hay más solución que intercalar STOP en los lugares clave y hacer uso de CONT para continuar el programa.

Tratamiento de ficheros

El ZX81 no permite el tratamiento de ficheros, pero es posible efectuar un pseudotratamiento mediante la utilización de cadenas y variables numéricas dimensionadas.

Vaya como ejemplo el siguiente programa que actúa como una agenda (este programa es demasiado grande para utilizarlo con un K).

```
  1 DIM A$(190,25)
  2 DIM D$(190,25)
  3 DIM T$(190,7)
  4 LET I=0
 10 CLS
 20 PRINT AT 2,11;"AGENDA";AT 1
0,4;"1.-INTRODUCCION DE DATOS";A
T 12,4;"2.-OBTENCION DE DATOS";A
```

```
T 14,4;"3.-GRABACION";AT 21,0;"O
PCION SELECCIONADA ?"
  30 IF INKEY$="" THEN GOTO 30
  40 LET R$=INKEY$
  50 IF R$<>"1" AND R$<>"2" AND
R$<>"3" THEN GOTO 30
  60 GOTO (90 AND R$="1")+(1000
AND R$="2")+(7000 AND R$="3")
 100 LET I=I+1
 110 GOSUB 5000
 115 PRINT AT 3,10;I
 120 INPUT A$(I)
 140 PRINT AT 8,4;A$(I)
 150 INPUT D$(I)
 160 PRINT AT 13,4;D$(I)
 170 INPUT T$(I)
 180 PRINT AT 16,12;T$(I)
 190 LET S=0
 195 PRINT AT 21,0;"CORRECTO?"
 198 PAUSE 100
 200 IF INKEY$="" THEN GOTO 200
 205 LET R$=INKEY$
 210 IF R$<"S" THEN GOTO 120
 220 PRINT AT 21,0;"QUIERE ENTRA
R OTRO REGISTRO ?"
 225 PAUSE 100
 230 IF INKEY$="" THEN GOTO 230
 235 LET R$=INKEY$
 240 IF R$>"N" THEN GOTO 100
 245 LET L=I
 250 GOTO 10
1000 CLS
1005 DIM A(L)
1007 LET J=0
1010 PRINT AT 10,5;"QUE LETRA QU
IERE ?"
1030 INPUT L$
1040 FAST
1050 FOR K=1 TO L
1070 IF A$(K,LEN L$)=L$ THEN LET
 J=J+1
1075 IF A$(K,LEN L$)=L$ THEN LET
 A(J)=K
1080 NEXT K
1090 SLOW
1100 LET S=0
2000 GOSUB 5000
2005 LET S=S+1
2010 PRINT AT 3,10;A(S);AT 8,4;A
$(A(S));AT 13,4;D$(A(S));AT 16,1
2;T$(A(S));AT 21,0;"OTRO REGISTR
O ?"
2015 PRINT AT 0,10;"ULTIMO REGIS
TRO" AND S=J
2020 IF INKEY$="" THEN GOTO 2020
2030 LET R$=INKEY$
2040 IF R$>"N" AND S<J THEN GOTO
 2000
2050 GOTO 10
```

```
5000 CLS
5005 PRINT AT 3,2;"CODIGO:";AT 6
,2;"NOMBRE:";AT 7,2;"▄▄▄▄▄▄▄▄▄▄▄";AT
11,2;"DIRECCION:";AT 12,2;"▄▄▄▄
▄▄▄▄";AT 16,2;"TELEFONO:";AT 17,
2;"▄▄▄▄▄▄▄▄▄▄▄"
5010 RETURN
7000 CLS
7010 PRINT AT 10,3;"NOMBRE EN CI
NTA:";
7020 INPUT S$
7030 PRINT S$
7040 PRINT AT 13,7;"█▀▀▀▀▀▀▀▀▀▀▀▀▀
▀▀▀▀▀█";AT 14,7;"█PULSE UNA TECLA
█";AT 15,7;"█▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄█"
7050 IF INKEY$="" THEN GOTO 7050
7060 SAVE S$
7070 GOTO 10
```

READ, DATA y RESTORE

Estas tres instrucciones permiten la introducción de tablas de datos y no existen en el ZX81. Actúan del siguiente modo: por ejemplo, para cargar una matriz numérica (A) con valores del 1 al 10 se escribiría el siguiente programa:

```
10 DIM A (10)
20 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.
30 FOR I = 1 TO 10.
40 READ A(I)
50 NEXT I.
```

Este programa actuaría cargando la matriz con los valores del 1 al 10. Si en algún punto se introduce la instrucción RESTORE, la próxima instrucción READ leerá el primer dato de DATA. Esto se puede sustituir por el siguiente programa que consiste en una subrutina para cada instrucción y en el que hay que poner los datos en sentencias REM separados por comas.

Si se usan varias sentencias REM al final de cada una hay que poner /, excepto en la última que hay que poner *, aunque estos símbolos se pueden cambiar a voluntad a la vista del programa:

```
9000 REM SUBRUTINA READ
9005 LET A$=""
9010 LET C$=CHR$ PEEK I
9020 IF C$="," THEN LET I=I+1
```

```
9030 IF C$="," THEN RETURN
9040 IF C$="/" THEN LET I=I+7
9050 IF C$="/" THEN RETURN
9060 IF C$="*" THEN LET I=16514
9070 IF C$="*" THEN RETURN
9080 LET A$=A$+C$
9085 LET I=I+1
9090 GOTO 9010
9500 REM SUBRUTINA RESTORE
9510 LET I=16514
9520 RETURN
```

La variable I contiene la dirección en que se encuentran los datos del REM que se supone que están al principio del programa; en caso contrario, hay que cambiar las líneas 9060 y 9510, asignando a I la nueva dirección del principio de los datos. Para utilizarlo pruebe con el siguiente ejemplo:

```
  10 REM LUNES,MARTES,MIERCOLES/
  20 REM JUEVES,VIERNES,SABADO,D
OMINGO*
  25 LET I=16514
  30 DIM S$(7,9)
  40 FOR K=1 TO 7
  50 GOSUB 9000
  60 LET S$(K)=A$
  70 PRINT S$(K)
  80 NEXT K
```

A las cadenas S$ (1) ... S$ (7) se les asignan los nombres que están en las sentencias REM y se imprimen seguidamente. No se olvide de poner la instrucción que asigna a I la dirección del principio de los datos (en este caso está en la línea 25).

Este programa actúa como base de datos haciendo la función de una agenda. Las sentencias DIM que se encuentran al principio están preparadas para 16 K pero pueden ser más grandes si se dispone de más memoria.

DIM A$(190,25)
DIM D$(190,25)
DIM T$(190,70)
LETI = 0

Con un K de RAM no cabe el programa; con 16 K se pueden tener perfectamente 190 registros en la agenda y por cada 16 K, 280 registros más.

El programa es muy fácil de hacer funcionar y se explica por sí mismo. Consta de un menú con tres opciones (líneas 10 a 60) que son: introducción de datos, obtención de datos, y grabación.

En la cadena A$ se almacenan los nombres, en D$ las direcciones (ambos previstos de 25 caracteres como máximo) y en T$ los teléfonos, de 7 caracteres. Estas dimensiones se pueden variar teniendo en cuenta el gasto de memoria y la impresión por pantalla. Como ejercicio, el lector puede añadir más opciones, como por ejemplo un listado por la impresora. En la impresión por pantalla se imprime también un código que corresponde a la primera dimensión de las cadenas, por si se quiere eliminar algún registro, ya que no se ha incluido como opción.

La introducción de datos se desarrolla en las líneas 100 a 250, la obtención de datos en las líneas 1000 a 2050 y selecciona únicamente aquellos que empiezan por la letra o cadena que se introduce (se compara con el nombre). Cuando sale por pantalla el último registro seleccionado aparece un aviso (línea 2015). Las líneas 5000 a 5010 forman la rutina de impresión, y las líneas 7000 en adelante la de grabación en cassette.

Es de observar que cuando se pide una respuesta se hace mediante INKEY$, como en las líneas 30, 40, 50, 60 que piden la opción deseada. Esto evita el tener que pulsar NEW LINE.

IF... THEN... ELSE... Esta instrucción actúa igual que IF... THEN, excepto en que la sentencia que sigue a ELSE se ejecutará en caso de que no se cumpla la condición.

Esta instrucción es difícil de sustituir, pero se puede hacer en algunos casos utilizando los operadores lógicos, por ejemplo IF B$ = "SI" THEN LET A=4 ELSE LET A=8

Se puede sustituir por LET A=(4 AND B$=<>"SI" + (8 AND BS< >«SI»).

Esta misma idea es la que nos sirvió en su momento para sustituir la instrucción ON GOTO y también servirá para ahorrar memoria en otras sentencias. Bastará repasar el Capítulo dedicado a operadores lógicos para comprender exactamente su funcionamiento.

CAPITULO 20

CODIGOS DE ERROR

En este Capítulo se comentan uno por uno todos los códigos de error que pueden aparecer al ejecutarse un programa con el ZX81.

0 .... Aparece cuando una orden se ejecuta correctamente o bien cuando se termina un programa debido a la ejecución de la última línea (la de número más alto).

1 .... Aparece cuando no existe la variable de control que sigue a la instrucción NEXT pero, en cambio, sí existe una variable ordinaria con el mismo nombre.

2 .... Indica que se ha utilizado una variable que no ha sido definida previamente. Es decir, aparece en una variable simple (ya sea numérica o de cadena) si no ha sido asignada previamente mediante una instrucción LET o INPUT. Las variables dimensionadas no quedan definidas hasta que se dimensionan con DIM. También se da cuando no existe una variable de control pero tampoco una variable simple con el mismo nombre.

3 .... Ocurre cuando el subíndice de una variable dimensionada se sale de los límites especificados en la sentencia DIM. También se da este error cuando el segmentado de una cadena no concuerda con su longitud o bien uno de los números toma el valor 0.

4 .... Se da cuando no hay sitio en la memoria para ejecutar una instrucción. Debido a esto, puede darse el caso de que quede tan poca memoria que el número de línea que le sigue esté incompleto o ni siquiera aparezca. Puede ocurrir en la ejecución de cualquier instrucción que pro-

duzca un gasto de memoria, por ejemplo DIM, LET, INPUT, FOR, etc....

5 .... Aparece cuando no hay sitio en la pantalla para imprimir o bien para listar un programa. En el caso de PRINT se puede usar CONT; pero en el caso de LIST, CONT produce el mismo listado.

6 .... Indica lo que se llama OVERFLOW; es decir, que en algún cálculo ha resultado un número más grande que el máximo almacenable en la memoria del ZX81 (aproximadamente 10^{38}). Lo más corriente es que se haya producido una división por un número muy cercano a 0.

7 .... Indica que no hay GOSUB para una sentencia RETURN, es decir que al pasar por la sentencia RETURN no hay ningún número almacenado en la pila del GOSUB.

8 .... Aparece cuando se intenta utilizar INPUT como una orden.

9 .... Indica que el programa se ha detenido por una sentencia STOP.

A .... Indica que el argumento de las funciones no es correcto. También aparece cuando se intenta elevar a cualquier potencia un número negativo.

B .... Muchas instrucciones utilizan números enteros. Si éstos se salen de escala aparece el error B. La escala para números enteros de un byte es 0—255 y son utilizados en CHR$ y en el segundo parámetro de POKE. Los de dos bytes están en la serie 0—65535 y se utilizan en todos los demás casos: RUN, RAND, PEEK, PAUSE, GOTO, GOSUB, etc.

C .... Aparece cuando la cadena a la que se aplica VAL no forma una expresión numérica. Es decir, contendría errores de sintaxis si formara parte de un PRINT.

D .... Se da cuando el programa se interrumpe con BREAK o bien cuando se utiliza STOP en INPUT.

E .... Este código no se usa, pero se puede hacer aparecer como se explica en el Capítulo dedicado a las variables del sistema.

F .... Se ha intentando cargar un programa dando, como nombre, la cadena vacía.

APENDICE: EL ZX-SPECTRUM

Cerca de un año después de sorprendernos con el lanzamiento del ZX81, Clive Sinclair lo consigue de nuevo lanzando un nuevo modelo, el ZX-SPECTRUM. Básicamente consiste en un ZX81 mejorado, con color, sonido, teclado móvil, gráficos de alta resolución, caracteres gráficos definibles por el usuario, minúsculas, un sistema de conexión a cassette notablemente mejorado y mucho más rápido, un lenguaje BASIC considerablemente ampliado y unas extraordinariamente atractivas ampliaciones: el interfaz RS232 y Net para interconexión de ZX-SPECTRUMs y los MICRODRIVES, dispositivos de almacenamiento rápido y masivo de memoria, de los que se pueden conectar hasta 8 a un sólo ZX-SPECTRUM y con una capacidad de hasta 100 Kbytes cada uno.

El BASIC del ZX-SPECTRUM es prácticamente el mismo que en el ZX81 salvo algunas excepciones que se mencionan a continuación, de modo que la mayoría de programas listados en BASIC del ZX81 podrán ser tecleados directamente en el ZX-SPECTRUM sin ninguna o muy pocas modificaciones.

Por desgracia, y debido al sistema de codificación y velocidad de transmisión distintos de datos en cassette, los programas del ZX81 grabados en cinta no podrán ser cargados en el ZX-SPECTRUM, a menos que se disponga de un programa adaptador que ya existe en el mercado. Y por último, los programas en código máquina serán en la mayoría de los casos totalmente incompatibles, a menos que no interfieran para nada con la gestión de pantalla y teclado, variables de sistemas y vías de acceso («port») de entrada/salida (MIC/EAR).

Las instrucciones del ZX81 que no existen en el ZX-SPECTRUM son las siguientes:

— **FAST y SLOW,** ya que el ZX-SPECTRUM trabaja siempre a la misma velocidad y con la pantalla siempre visible. De modo que

las líneas de los listados de programas del ZX81 en que aparezcan estas instrucciones deberán ser eliminadas.

- **SCROLL.** El ZX-SPECTRUM tiene un sistema de «enrollamiento» automático de la pantalla, preguntando **Scroll?** cada vez que se llena. Si se aprieta cualquier tecla que no sea STOP, SPACE ó N, el ZX-SPECTRUM presenta automáticamente otra pantalla; en caso contrario, se para. Sin embargo, se puede sustituir la instrucción SCROLL del ZX81 en el ZX-SPECTRUM y con el mismo efecto, con: IF USR 3582 THEN PRINT AT 21,0.

- **UNPLOT.** Debe ser sustituido por PLOT OVER, cuya función es ligeramente distinta y se explicará más adelante. En los programas que utilicen los gráficos hay que tener en cuenta las distintas resoluciones (64 × 44 puntos en el ZX81 y 256 × 178 en el ZX-SPECTRUM) y hacer las consiguientes funciones de corrección y adaptación: por ejemplo, simplemente multiplicar por 4 los valores del ZX81.

La conexión del ZX-SPECTRUM se realiza del mismo modo que la del ZX81, ya que se suministra con los mismos cables para televisor y cassette y un alimentador del mismo tipo, aunque más potente.

Otra importante diferencia entre el ZX81 y el ZX-SPECTRUM es el sistema adoptado en la codificación de los caracteres y códigos de control. Así, mientras el ZX81 usa un código totalmente propio que lo hacía incompatible para la fácil conexión a impresoras normales y otro tipo de periféricos, el ZX-SPECTRUM utiliza el código normalizado ASCII, con lo que resulta mucho más sencillo conectarlo a periféricos de otras marcas.

EL TECLADO

El teclado del ZX-SPECTRUM está compuesto, al igual que el del ZX81, por 40 teclas pero con la diferencia de que son móviles, lo que proporciona una mayor sensación de pulsación que, junto con el avisador acústico de pulsación, hace mucho más cómodo teclear en el ZX-SPECTRUM que en el ZX81. Otra diferencia notable es el número de funciones por tecla, que en el ZX-SPECTRUM pueden ser hasta 8. Por ejemplo, de la tecla I se pueden obtener mediante las pulsaciones adecuadas de la misma y otras teclas de control:

- La I mayúscula.

— La i minúscula.
— La palabra clave INPUT
— La función AT.
— La función IN.
— La función CODE.
— Un gráfico definible por el usuario.
— Y además todos los anteriores en video inverso y en cualquier combinación de colores de fondo y del carácter y con atributos de brillo y/o parpadeo.

La tecla NEWLINE del ZX81 está sustituida en el ZX-SPECTRUM por ENTER, que es la única tecla que tiene sólo una función. Pulsando en las combinaciones y orden adecuadas las teclas SYMBOL SHIFT y CAPS SHIFT se obtienen todas las funciones de las teclas. En la primera línea del teclado, en lugar de los números en «mayúsculas» se obtienen los códigos de control de los colores, gráficos y video inverso.

CARACTERISTICAS ADICIONALES DEL ZX-SPECTRUM SOBRE EL ZX81

1. Color

Esta es, sin duda, la característica más sobresaliente del ZX-SPECTRUM frente al ZX81. El ZX-SPECTRUM tiene una salida de señal de TV en colores, aunque sigue siendo posible conectarlo a un televisor en blanco y negro en el que se obtendrán distintos tonos de grises en lugar de los colores.

Para aprovechar las extraordinarias posibilidades que el color brinda al ordenador, se han introducido un conjunto de instrucciones y comandos BASIC, que se describen a continuación:

La pantalla del ZX-SPECTRUM está dividida en dos secciones, que podemos llamar BORDE y PAPEL (BORDER y PAPER en inglés). La primera rodea a la segunda, que es donde se visualizarán las 24 líneas de 32 caracteres. Las instrucciones **BORDER** y **PAPER** se utilizan para asignar el color a cada una de estas secciones, siendo los posibles colores los siguientes:
0 negro
1 azul
2 rojo
3 magenta

4 verde
5 cyan
6 amarillo
7 blanco
y se utilizan, por ejemplo, de este modo:

BORDER 0 ENTER
PAPER 6 ENTER CLS ENTER

Podemos pues observar, que PAPER no tiene efecto hasta que se hace CLS, con lo que cada vez que se quiera cambiar globalmente el color de PAPER, se borra todo el contenido de la pantalla.

Ambas instrucciones pueden usarse, por descontado, dentro de un programa.

Las 24 líneas de 32 caracteres de la pantalla pueden ser consideradas como 768 posiciones de carácter en total, de las que 22 × 32 son normalmente accesibles por el usuario. Cada posición de carácter puede tener unas características o atributos que pueden ser especificadas por el usuario. Estas son:

— color del fondo (PAPER de cada carácter).
— color del carácter en sí (INK o tinta), que si fuese el mismo que PAPER no se distinguiría dicho carácter.
— brillo, que se asigna con la instrucción BRIGHT 0 (brillo normal) o BRIGHT 1 (brillo extra).
— intermitencia, asignado por FLASH 1 (sí intermitencia) y FLASH 0 (no intermitencia), que alternan los colores de INK y PAPER un par de veces por segundo.

Además de los 7 colores descritos, puede utilizarse el 8 que significa **transparencia**, es decir, dejarlo como estaba, y el 9 que significa **contraste.** PAPER será negro si INK es un color claro (0 a 3) y blanco si INK es un color oscuro (4 a 7). El color 9 puede usarse también del modo inverso, es decir con INK teniendo en cuenta el PAPEL actual, pero no puede emplearse con BORDER.

Hay otras dos sentencias que afectan el contenido de una posición de carácter: **INVERSE 1** la pone en video inverso e **INVERSE 0** en vídeo normal;

OVER 1 permite que un carácter asignado a una posición de carácter que ya contiene uno, se sobre-escriba sobre éste sin borrarlo, mientras que **OVER 0** hace que se borre el anterior.

Considerando que cada posición de carácter está formada por una matriz de 8 × 8 puntos o pixels, cada uno de los cuales puede ser de color INK o PAPER, el efecto de OVER no es totalmente la su-

perposición absoluta, sino que donde hay un pixel INK y tiene que superponer otro INK, en realidad lo borra y lo deja PAPER, mientras que donde hay un INK del carácter anterior o un INK del carácter nuevo que no interfieren entre ellos, se quedan INK. Es decir, realiza una función lógica OR-exclusiva entre los pixels de ambos caracteres que se superponen.

Tanto INK, PAPER, FLASH, BRIGHT, INVERSE y OVER pueden utilizarse en conjunción con instrucciones PRINT e INPUT y con instrucciones gráficas como DRAW y PLOT.

Por ejemplo:

PRINT INK 3; FLASH 1; "HOLA, BUENOS DIAS"
INPUT PAPER 4; BRIGHT 1; "Qué día es hoy?"; D$

La función **ATTR** devuelve un número que codifica todos los atributos posibles de una posición de carácter. Se usa de este modo, por ejemplo: PRINT ATTR (10 12). Y el número que devuelve es la suma de las siguientes cantidades:

FLASH 1 = 128
FLASH 0 = 0
BRIGHT 1 = 64
BRIGHT 0 = 0
PAPER n = 8*n
INK n = n

Así, si ATTR (10,12) = 231, la posición de carácter 10,12 tendrá los siguientes atributos:

FLASH 1 = 128
BRIGHT 1 = 64
PAPER 4 = 8*4 = 32
INK 7 = 7
128 + 64 + 32 + 7 = 231

El efecto de INVERSE 1 e INVERSE 0 también puede obtenerse con las teclas INV, VIDEO y TRUE VIDEO respectivamente. El efecto real de INVERSE 1 es invertir los colores de INK y PAPER, mientras que INVERSE 0 los deja normales.

2. Sonido

La instrucción **BEEP n,m** produce un sonido de duración n segundos y tono m semitonos por encima del Do central. Dando valores decimales pequeños pueden obtenerse variaciones del tono y duraciones pequeñas que permitan conseguir distintos efectos de sonido.

3. Separación de instrucciones en una misma línea

Una gran ventaja del BASIC del ZX-SPECTRUM sobre el del ZX81 es la posibilidad de colocar varias instrucciones en una misma línea separadas por dos puntos (:). De este modo se puede ahorrar espacio de memoria notablemente y resulta bastante más cómodo programar. Por ejemplo:

```
FOR a= 1 TO 100: BEEP .5, a: NEXT a
LET a = 1: LET b = 3: LET c = 6: DIM A (40,50)
```

Hay que tener en cuenta, sin embargo, que el abuso de esta posibilidad puede complicar la estructura de los programas haciéndolos incomprensibles.

Si la primera instrucción de una línea es un GOTO, el programa ya no pasará por las instrucciones del resto de la línea. Esta característica hay que tenerla en cuenta también en las sentencias IF...THEN. Si se cumple la condición, el programa seguirá ejecutando lo que le venga a continuación en la misma línea, pero si no se cumple saltará automáticamente a la línea siguiente.

4. Instrucciones y comandos BASIC

— La instrucción RAND del ZX81, aparece del mismo modo en el teclado del ZX-SPECTRUM, pero como RANDOMISE en la pantalla.
— El símbolo ** del ZX81 para elevar a una potencia, se representa por ↑ en el ZX-SPECTRUM.
— El funcionamiento de CLEAR en el ZX-SPECTRUM es distinto que en el ZX81, ya que no sólo borra todas las variables presentes en memoria, sino que además reinicializa la variable de sistema RAMTOP, hace RESTORE y borra el stack de GOSUBs. Puede también emplearse para reservar una zona de memoria protegida por encima de RAMTOP: usando el comando CLEAR n, siendo n por ejemplo 30000, RAMTOP se fija en 30000.
— Se han incorporado una serie de características adicionales a la sentencia INPUT en el ZX-SPECTRUM. Es posible entrar varias variables en una sola línea. Por ejemplo:
INPUT A, B, C, D
Es posible incluir un texto para que aparezca en las dos líneas de abajo de la pantalla al hacer un INPUT. Por ejemplo:

INPUT " Cómo te llamas?", N$

Es posible hacer INPUT AT n,m;X$ para hacer INPUT en determinadas líneas de la pantalla.

INPUT LINE X$ permite hacer un INPUT de una variable alfanumérica sin que aparezcan las comillas a ambos lados del curso.

— La función VAL$ del ZX-SPECTRUM que, aparentemente, no tiene ninguna utilidad, se dice que evalúa una cadena alfanumérica y la deja como otra cadena alfanumérica. Un ejemplo:

VAL$ """"SPECTRUM"""" = "SPECTRUM"

Parece que no sirve más que para eliminar comillas (??).

— El conjunto de instrucciones READ-DATA-RESTORE que tanto se echaba a faltar en el ZX81 ha sido, por fortuna, implementado en el ZX-SPECTRUM. Permiten almacenar y disponer de los datos de las mismas líneas del propio programa. Por ejemplo:

```
10 READ a,a$
20 PRINT a,a$
30 DATA 80, "PEPE"
```

Este pequeño programa asigna a = 80 y a$ = "PEPE". Nótese que:

* Los datos se colocan en líneas DATA que pueden contener tantos como sea posible en una línea de programa, estando separados por comas y que pueden ser de distintos tipos, teniendo que estar entre comillas los alfanuméricos.

* Las variables en sentencias READ deben estar separadas por comas.

Pueden ponerse varias líneas DATA seguidas o en distintas partes del programa. Al ejecutarse READ se empezará a leer siempre la primera línea DATA del programa, a menos que se haga RESTORE n, donde n es el número de línea DATA a partir de la cual se quiere empezar a hacer READ. RESTORE sólo vuelve a iniciar el READ desde la primera línea DATA.

— El BASIC del ZX-SPECTRUM proporciona también al usuario la posibilidad de definir funciones numéricas y de cadenas de caracteres. Hay 26 de cada definibles. Las numéricas se designan con FN seguida de una letra mayúscula (por ejemplo: FN A). Y las cadenas de caracteres por FN, seguido de una letra mayúscula y el signo de dólar. Por ejemplo: FN A$. Las funciones se definen con la sentencia DEF FN. Por ejemplo:

DEF FN A(x,y,z) = SQR x + (4*y) + z

DEF FN A$ (B$) = B$ (2 TO 6)

Y si damos posibles valores a x, y, z y B$ mediante FN:

FN A(4,3,2) = SQR 4 + (4*3) + 2 = 16

FN A$("ZX-SPECTRUM") = "X-SPE"
Es decir si hacemos:
PRINT FN A(4,3,2)
PRINT FN A$("ZX-SPECTRUM")
obtendremos 16 y "X-SPE" respectivamente y estas funciones definidas nos evaluarán las mismas con cualquier valor que les coloquemos.

— El ZX-SPECTRUM dispone también de la sentencia OUT y la función IN que se utilizan respectivamente para escribir y leer en las vías de acceso («port») de entrada/salida del microprocesador Z80, que podemos definir como las líneas que le comunican con sus periféricos, que podrían ser por ejemplo el teclado, el cassette, el televisor, etc.
Asi OUT se utiliza de la forma OUT m,n donde m es el número de la vía de acceso y n el número que se escribe en él. Por ejemplo, la vía de acceso 254 es la salida del altavoz y la conexión de micrófono, y asigna al mismo tiempo el color de BORDER. Para leer, por ejemplo, la sección CAPS SHIT a V del teclado, haremos:
PRINT IN 65278. Otras direcciones útiles se describen en el manual del ZX-SPECTRUM.

— La función SCREEN$ (x,y) proporciona el carácter que está (aunque sea en video inverso) en las coordenadas (x,y) de la pantalla. Si no identifica el carácter, caso de los caracteres definidos por el usuario devuelve la cadena vacía.
Por ejemplo:
PRINT AT 12, 12;"*"
PRINT SCREEN$ (12, 12) dará "*"

5. Gráficos de alta resolución

En el ZX-SPECTRUM es posible «dibujar en trazo fino» en la pantalla con la resolución de un pixel, que es, como hemos dicho antes, un punto de la matriz de 8 × 8 que configuran una posición de carácter. Esta es la causa por la que la pantalla del ZX-SPECTRUM ocupa en memoria 6912 bytes, por lo que un ZX-SPECTRUM de 16K tiene solamente unos 9K realmente disponibles por el usuario, y uno de 48K, unos 41K reales.

Se dispone de una serie de instrucciones para la realización de gráficos en alta resolución, que son:
PLOT, DRAW, CIRCLE y POINT

Al igual que en el ZX81, PLOT coloca un punto en las coorde-

nadas especificadas de la pantalla, que pueden ser de 0 a 255 en horizontal y de 0 a 175 en vertical, empezando por la esquina inferior izquierda de la línea 22 de la pantalla, ya que no se pueden utilizar las instrucciones de alta resolución de las dos líneas inferiores.

DRAW x,y traza una línea recta tomando como origen de coordenadas la última coordenada utilizada y x,y como las coordenadas de destino a partir del nuevo origen, por lo que es posible que x e y, al ser coordenadas relativas, puedan tomar valores negativos. Por ejemplo:
PLOT 50,50: DRAW 45,40
trazará una línea recta desde las coordenadas absolutas 50,50 hasta la 50 + 45,50 + 40, es decir 95,90.

Se puede asignar un tercer parámetro tal como DRAW x,y,z donde z es el arco de circunferencia en radianes que ha de describir el trazado para ir de las coordenadas iniciales a las finales. Se utiliza para el trazado de arcos y para conseguir ciertos «efectos especiales» gracias al sistema con que está implementada esta instrucción. Probar por ejemplo con:
PLOT 50,50: DRAW 50,80,200000000 y con otros valores muy grandes para el arco.
CIRCLE x,y,z traza una circunferencia con centro en las coordenadas absolutas x,y y radio z.

Los atributos de INK, INVERSE y OVER pueden también utilizarse en alta resolución. Por ejemplo:
PLOT OVER 1; 23,23
DRAW INVERSE 1;56,67

La función POINT (x,y) devuelve un 0 cuando el pixel (x,y) es de color PAPER en la determinada posición de carácter en la que se encuentre y un 1 cuando es de color INK.

6. Almacenamiento de programas y datos en cassette

Este es un apartado que ha sido notablemente mejorado en el ZX-SPECTRUM respecto al ZX81. De entrada, la velocidad de transmisión es seis veces superior, 1500 baudios o bits por segundo en lugar de los 250 del ZX81, lo que significa en principio que un programa del mismo número de K tarda seis veces menos en grabarse o en cargarse. Teniendo en cuenta que 1K son 1024 bytes y que cada byte está compuesto de 8 bits, mediante una simple operación

aritmética se puede calcular lo que tardará exactamente un programa en cargarse, si se sabe exactamente los K que ocupa.

Todos los programas o datos se graban en dos partes separadas, una conteniendo el nombre y datos del mismo, y a continuación el programa o datos o zona de memoria en sí, ambos precedidos por un tono fijo de algunos segundos de duración, que sirve para estabilizar los grabadores con control automático de grabación. Para almacenar programas, datos o zonas de memoria en cinta, se utiliza la instrucción SAVE, que puede tener los siguientes formatos:

SAVE «nombre»: almacena el programa junto con todas sus variables.

SAVE «nombre» LINE n: almacena el programa junto con las variables, de modo que al cargarse nuevamente en el ZX-SPECTRUM se pone en marcha automáticamente a partir de la línea n.

SAVE «nombre» LINE: almacena el programa junto con las variables, de modo que al cargarse de nuevo en el ZX-SPECTRUM se pone en marcha automáticamente a partir de la primera línea.

SAVE «nombre» DATA x(): almacena la matriz o tabla numérica x con el nombre especificado.

SAVE «nombre» DATA x$(): almacena la matriz o tabla alfanumérica x$ con el nombre especificado.

SAVE «nombre» CODE m,n: almacena los n bytes de memoria desde el número m, es decir desde m hasta m + n.

SAVE «nombre» SCREEN$: almacena los 6912 bytes que constituyen la memoria de pantalla o archivo de imagen.

Para verificar la correcta grabación de los datos o programas, se rebobina la cinta y se procede como si se cargase el programa en memoria utilizando VERIFY.

VERIFY «nombre»
VERIFY «nombre» DATA x()
VERIFY «nombre» DATA x$()
ó
VERIFY «nombre» CODE m,n.

Un caso particular es VERIFY «nombre» SCREEN$ que dará siempre error, ya que lo que había en la pantalla al hacer SAVE no es lo mismo que al hacer VERIFY, puesto que las propias palabras SAVE ó VERIFY de la instrucción están en la pantalla.

Para cargar programas o datos en el ZX-SPECTRUM se utiliza LOAD en uno de los siguientes formatos:

LOAD "" ó LOAD «nombre», que no cargarán DATA ni CODE.

LOAD "" DATA x() ó LOAD «nombre» DATA x()
LOAD "" DATA x$() ó LOAD «nombre» DATA x$()
LOAD ""CODE o LOAD «nombre» CODE
LOAD "" SCREEN$ ó LOAD «nombre» SCREEN$
LOAD "" CODE m,n ó LOAD «nombre» CODE m,n puede utilizarse para cargar unos bytes de memoria en una zona distinta de la que han sido grabados en cinta previamente.

También es posible la mezcla de programas BASIC con MERGE " " ó MERGE «nombre». El segundo programa que se MERGE con el primero siempre domina sobre éste, de modo que si contiene líneas de programa con el mismo número, se quedan las del segundo programa, y de otro modo se complementan.
Puede utilizarse MERGE para cargar programas que normalmente se ponen en marcha automáticamente sin que esto ocurra, aunque puede ocurrir que algunos programas estén incluso protegidos contra esto.

En cualquier caso, si al cargar un programa se produjese un error de lectura o de falta de memoria, lo que ya se haya cargado no quedará destruido como en el ZX81, excepto en casos bastante aislados.

8. Juego de caracteres

Como se ha dicho anteriormente, el ZX-SPECTRUM utiliza el código de caracteres normalizado ASCII, por lo que en el teclado se encuentran las minúsculas y otros caracteres especiales que no estaban en el ZX81, tales como (c), &, ', !, # y otros.

Se tiene además la posibilidad de definir 21 caracteres gráficos correspondientes a las teclas A a U. Sabiendo que una posición de carácter está formada por una matriz de 8 × 8 puntos, vemos que tenemos la posibilidad de definir cada uno de esos 64 puntos para crear nuestros propios caracteres. Cada uno de estos puntos puede estar «encendido» o «apagado», o sea que puede ser representado por un bit 1 ó un bit 0, y como cada 8 bits forman un byte, tenemos que un carácter está definido por 8 bytes consecutivos en memoria.

Pongamos una sencilla definición. Por ejemplo, un carácter de los grises que tiene el ZX81, pero que no está en el ZX-SPECTRUM, estaría formado por:
10101010

01010101
10101010
01010101
10101010
01010101
10101010
01010101

estos números binarios corresponden en decimal a:

170
85
170
85
170
85
170
85

Para definirlo en la tecla A, utilizamos la siguiente rutina:

```
10 FOR n = 0 TO 7
20 READ a
30 POKE USR "A" + n,a
40 NEXT n
50 DATA 170,85,170,85,170,85,170,85
```

Disponemos también de la función BIN que permite entrar números binarios directamente. Así, podríamos sustituir la línea 50 por:

```
50 DATA BIN 10101010, BIN 01010101, BIN 10101010, BIN 01010101,
BIN 10101010, BIN 01010101, BIN 10101010, BIN 01010101.
```

9. Mensajes de error

Los mensajes de error del ZX-SPECTRUM son mucho más claros que los del ZX81 ya que dan información legible sobre qué consiste el error además de su código correspondiente, la línea donde se encuentra el error y el número de orden de la instrucción de esa línea que contiene el error, si es una línea con varias instrucciones separadas por (:).

10. Microdrives

El ZX-SPECTRUM tiene también una serie de instrucciones y co-

mandos asignados a los microdrives. Dado que a la hora de escribir este libro, este equipo adicional todavía no está disponible, no ha sido posible explicar su funcionamiento.

Estas instrucciones y comandos son:
OPEN #, CLOSE #, MOVE, ERASE, CAT y FORMAT

11. Mapa de memoria

Una diferencia notable con el ZX81 es que el archivo de imagen permanece fijo, por lo cual resulta mucho más rápido y sencillo el acceso del mismo, aunque luego la cosa se complica por la manera en que está montado.



Guía práctica de BASIC del ZX-81 y del Spectrum

por RAMON ROVIRA S.